Gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

Ianuarie 2001
Vol. 11/1

Lei 15.000

din sumarul numărului:

10 ANI de Gazeta de informatica

BABEL
Măsurarea timpului

SERIAL
Structură și stil în programare

ACTUAL
DirectX 8.0

TEHNO Războiul procesoarelor

La multi ani!

PC MAGAZINE ROMÂNIA

Mesageria instantanee în sprijinul activității din firme

Un diagnostic autorizat asupra celor mai importante situri medicale

COMPUTER PRESS AGORA

Corel LINUX OS
HP Deskjet 970Cxi
PDA Psion Series 7
NetWare 4.5 beta
Compaq Presario
Gateway Profile 2cx
Xircom PortStation
HP CD-Writer Plus 9210i

PC Magazine a testat 12 acceleratoare grafice pentru mediul de afaceri,

Din acest număr:

- **Web mai rapid**
- **Internet business**
- **Windows și Linux**
- **Cadouri de sărbători**

www.pcmagazine.ro

Revista utilizatorilor de calculatoare personale

COMPUTER PRESS AGORA
Str. Tudor Vladimirescu nr. 63/1, Tg.Mureș (CP 230/1), 4300
Tel. 065-166516, Fax 065-166290, e-mail: office@agora.ro

Vol. 11, Nr. 1 /2001

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Redactor:
Carmen Șușman (Alba Iulia)

Au mai colaborat:
Laviniu Bejenaru (Dr. Tn. Severin)
Crina Groșan (Cluj)
Mihai Oltean (Cugir)
Bazil Pârv (Cluj)
Ionuț Stan (Aiud)
Radu Cristian Vișinescu (Ploiești)

Tehnoredactare:
Călin Borz

Coperta și grafica:
Adrian Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 064/192.422
065/166.516
Fax: 065/166.290
http://www.ginfo.ro

ISSN: 1222-7129
Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359
Editor: Computer Press Agora SRL
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș
Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihaela Szabo (mszabo@agora.ro)
Ioana Udroiu (iudroiu@agora.ro)
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

Σ

sumar

actual.. PAG. 5

DIRECTX 8.0

Carmen ȘUȘMAN

concurs.. PAG. 7

Rezultatele rundei a doua

Runda a patra

focus.. PAG. 21

Configurații ORDONATE

Mihai OLTEAN
Crina GROȘAN

Algoritmi de TARE-PARCURGERE

Radu Cristian VIȘINESCU

babel.. PAG. 29

Măsurarea TIMPULUI

Mihai SCORȚARU

Rezolvarea unei ECUAȚII cu coeficienți NUMERE întregi mari

Laviniu BEJENARU

tehno.. PAG. 35

Războiul PROCESOARELOR

Mihai SCORȚARU

serial.. PAG. 37

Structură și stil în PROGRAMARE

Bazil PÂRV

begin.. PAG. 43

Un nou mileniu... cu PROBLEME

Ionuț STAN

Primii pași în algoritmică

enjoy.. PAG. 48

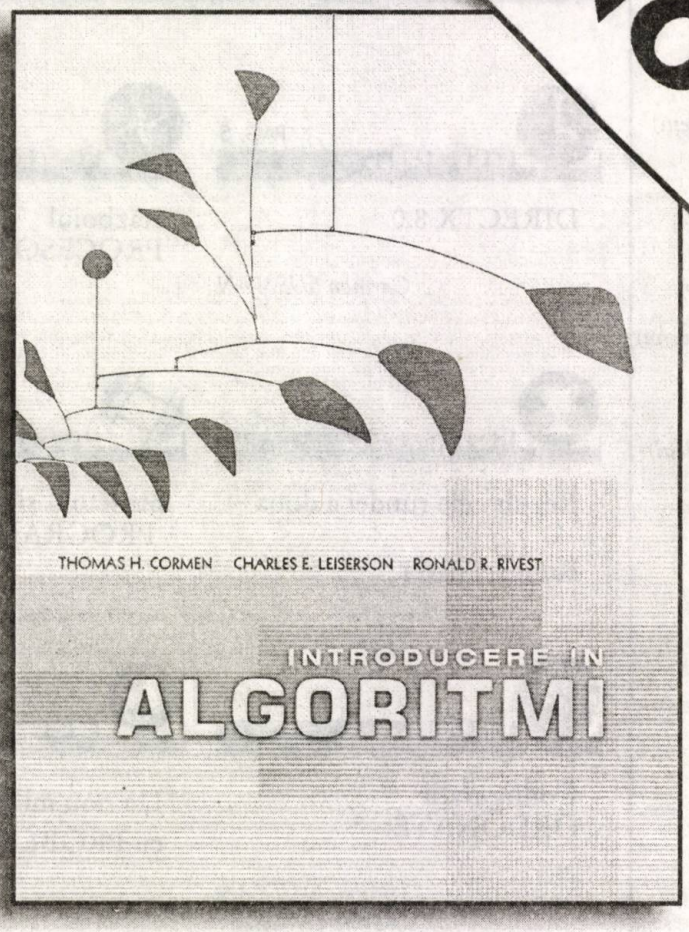
GM vs. MICROSOFT

Carmen ȘUȘMAN

Păreri despre industria automobilistică și cea producătoare de software

COMPUTER LIBRIS AGORA

NOUTATE



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

GInfo are 10 ANI!

Oare cum au trecut acești 10 ani? Parcă doar ieri (alaltăieri?) am propus editurii *Petrion*, împreună cu colegul *Adrian Atanasiu*, editarea unei reviste de informatică adresate elevilor și profesorilor. Modelul inițial a fost *Gazeta de Matematică*, revistă de tradiție și prestigiu; de aceea numele inițial al revistei a fost *"Gazeta de Informatică"*.

Startul a fost foarte greu datorită condițiilor precare de editare și mai ales datorită difuzării revistei. În plus, s-a simțit interesul redus și neîncrederea editorului în viitorul revistei. De aceea, după primele cinci numere (noiembrie 1991, martie 1992) entuziasmul nu era foarte mare. Editura *Petrion* a lansat *Gazeta de Informatică* pe piață într-un tiraj de 18.000 de exemplare, dar a reușit să vândă doar 2000.

După aceste cinci numere am avut norocul de a primi oferta *d-nei prof. Clara Ionescu* de a continua apariția revistei la editura *Libris*. A fost un moment important, consființind o colaborare de lungă durată și deosebit de fructuoasă. Entuziasmul și perseverența ei a făcut ca revista să ajungă în școli și să primim primele articole scrise de elevi și profesori. Așa cum spuneam, dificultățile de editare au fost foarte mari. Câțiva ani am redactat și tehnoredactat revista folosind editorul *WordPerfect* și alergând din loc în loc după o imprimantă laser. Alergătura nu s-a oprit aici: revista listată trebuia trimisă la Cluj unde se mai făcea o corectură, urmau discuții la telefon, un drum la gară etc. (noroc cu "nașu" care asigura o livrare rapidă). *Gazeta de Informatică* s-a tipărit la *Tipografia Ardealul* din Cluj, unde aproape că nu a existat număr la care să nu avem o surpriză (evident neplăcută), fie au schimbat culoarea copertei, fie au tipărit și însemnările tipografilor, fie au tipărit-o în alt tiraj decât era comandat.

Am menționat mai sus că de la început *Gazeta de Informatică* a vrut să fie pentru elevii care studiază informatica în școli, ceea ce era și este *Gazeta de Matematică* pentru cei care sunt pasionați de matematică. Pe parcursul anilor, pe paginile ei au fost publicate foarte multe articole valoroase din punct de vedere științific, cu o ținută elegantă, fără greșeli de conținut sau de formă, lucru la care au ținut mult atât redactorii cât și editura.

La fel de important a fost faptul că, încă de la început, *Gazeta de Informatică* a fost motorul pregătirii elevilor români pentru olimpiadele naționale și internaționale. Organizarea în premieră a *Balcaniadei de Informatică* și a *Olimpiadei de Informatică a Europei Centrale* a constituit o nouă sursă de probleme, o nouă șansă a elevilor români de a se antrena în condiții de concurs.

În 1994 am anunțat o pauză în apariția revistei, din cauza muncii enorme (și prost remunerate), a dezinteresului profesorilor de liceu (care au trimis foarte puține articole - și acelea în formă nefinisată - și care nu au pus în suficientă măsură umărul la difuzarea în școli a gazetei) și a altor nemulțumiri care nu este cazul să fie reamintite la această aniversare. În 1995 am avut un singur număr (Ediție Specială), iar în următorii doi ani câte trei numere în fiecare.

Din nou, motorul reapariției revistei, a fost dăruirea și entuziasmul *d-nei prof. Clara Ionescu*. Începând cu 1998 gazeta s-a renăscut din cenușă. Editura *Libris* s-a asociat cu editura *Computer Press Agora* din Târgu Mureș, editorul revistelor *PC Report* și *Byte*. A primit veșminte noi, și datorită viziunii inspirate a d-lui *Mircea Sârbu*, o structură nouă. Secțiunile de azi ale revistei sunt mai variate decât în anii 1991-1997 și revista are un subtitlu: ***revista viitorilor specialiști în tehnologia informației***. A crescut de la 32 de pagini și *WordPerfect* la 48 de pagini, tehnoredactate în *QuarkXPress*.

Azi nu mai mergem la gară, nu mai dăm telefoane îngrijorate, gazeta se trimite prin e-mail. Coperta este frumoasă, colorată, hârtia albă, avem reclame și sponsori.

GInfo, succesor demn al *Gazetei de Informatică* de odinioară, a fost în 1991 și a rămas și azi singura revistă de acest gen din lume. În vara anului 2000, a fost editat un număr special în limba engleză care a fost oferit participanților la *CEOI 2000*. Revista a fost și este unanim apreciată și a întărit părerea delegațiilor străine că ea constituie un factor important care contribuie la succesul elevilor români la concursurile internaționale de informatică.

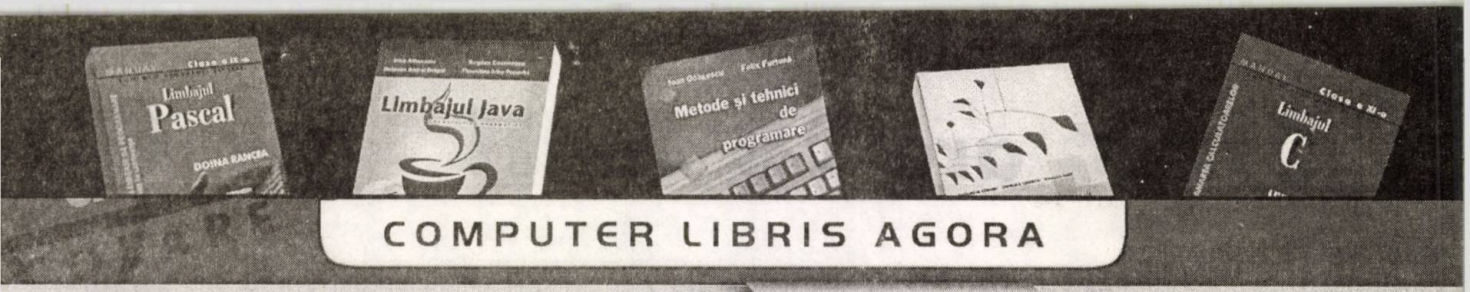
Un editorial de aniversare nu poate fi complet fără să amintesc articole care se citează și azi în cărți și alte reviste, fără să amintesc colaboratori fideli care pe parcursul anilor au publicat în paginile revistei. Dar lista ar fi prea lungă. Voi aminti doar autorii unor seriale cu succes: al lui *Mihai Scorțaru* despre *HTML*, al meu despre *limbajul Java*, al lui *Cristian Cadar* despre *algoritmi de teoria grafurilor*. Din păcate, continuă și retincența profesorilor de liceu în a trimite articole. Noroc cu foștii și actualii elevi care, prin articolele trimise, asigură material suficient pentru apariția revistei.

Nu pot să nu amintesc concursul de programare (inițiat și administrat de redacție) care, datorită talentului și ambiției lui *Mihai Scorțaru*, a devenit în 2000 concurs internațional cu aproape 400 de participanți.

Închei cu speranța că nu veți pierde interesul pentru *GInfo*, o veți citi și o veți populariza. Și încă ceva: îmi doresc ca, în viitor, atât elevii cât și profesorii să ne "bombardeze" cu materiale propuse pentru publicare.

La mulți ani, GInfo!

Horia Georgescu



COMPUTER LIBRIS AGORA

NOUTATE

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.

Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

DIRECTX 8.0

Carmen Șușman

La un an după apariția versiunii DirectX 7.0, Microsoft oferă cea de-a opta versiune a acestui produs. În acest articol vom prezenta informații generale despre DirectX, precum și câteva detalii referitoare la unele dintre noutățile introduse de cea de-a opta versiune.

Despre DirectX

Datorită faptului că firmele producătoare de hardware lansau pe piață tot mai des produse noi care nu erau compatibile unele cu altele, programatorii întâmpinau dificultăți tot mai mari în realizarea de programe care să nu depindă prea mult de configurația calculatorului pe care sunt rulate. Pentru a ușura sarcina programatorilor, compania *Microsoft* încearcă, prin intermediul pachetului *DirectX*, să realizeze o platformă virtuală stabilă, care să permită scrierea de programe, care să depindă cât mai puțin de configurația hardware.

Să presupunem că dorim să scriem un program care folosește accelerarea 3D ale plăcilor video. În funcție de placa video a calculatorului pe care va rula programul, instrucțiunile din codul sursă ar putea fi diferite. În *DirectX* este inclusă o funcție care permite folosirea accelerării 3D pentru toate plăcile video suportate. Funcția *DirectX* va "ști" care este placa video a calculatorului și va executa instrucțiunile corespunzătoare plăcii respective. În mod normal, programatorul ar fi trebuit să scrie codul sursă pentru toate plăcile video, dar așa, cei de la *Microsoft*, au scris codul sursă în locul său, iar el nu trebuie decât să apeleze o funcție *DirectX*.

Majoritatea jocurilor și aplicațiilor multimedia apărute în ultimul timp folosesc avantajele oferite de *DirectX*. Din acest motiv este aproape obligatoriu ca pe orice calculator să fie instalat acest produs. *DirectX* 8.0, la fel ca și versiunile anterioare, este gratuit și poate fi descărcat de pe siteul Web al companiei *Microsoft*.

În ultimul timp, *Microsoft* a integrat pachetul *DirectX* în sistemul de operare, ultimele variante de *Windows* incluzând și una dintre variantele *DirectX*. Totuși, acestea trebuie actualizate în momentul apariției unei noi versiuni, așa că acum e momentul pentru trecerea la *DirectX* 8.0.

Despre DirectX 8.0

Așa cum este normal, *DirectX* 8.0 păstrează compatibilitatea cu versiunile anterioare, deci orice aplicație care folosea versiunea *DirectX* 7.0 sau o versiune anterioară, va putea rula fără probleme după instalarea ultimei versiuni. Au apărut o serie de modificări în modul de implementare a vechilor funcții, care ar putea aduce chiar un spor de viteză dacă se trece la versiunea nouă. Totuși, în majoritatea aplicațiilor vechi, diferența nu va fi sesizabilă.

Au fost introduse o serie de noutăți semnificative, care fac din *DirectX* 8.0 un succesor demn al lui *DirectX* 7.0. O mulțime dintre noile aplicații vor fi scrise special pentru versiunea *DirectX* 8.0 și nu vor putea rula dacă este instalată o versiune anterioară.

DirectX 8.0 deschide o nouă cale pentru dezvoltarea aplicațiilor multimedia, în special a celor bazate pe rețeaua *Internet*. Noua versiune este disponibilă în variante pentru sistemele de operare pentru familia *Windows '9x* (*Windows '95*, *Windows '95 OSR 2*, *Windows '98*, *Windows '98 Second Edition* și *Windows Millennium*) și pentru familia *Windows 2000* (*Windows 2000 Professional* și *Windows 2000 Server*). Nu este oferit suport pentru *Windows NT* 4.0. Singura versiune *DirectX* care este recomandată pentru *WinNT 4.0* este *DirectX 3.0a*, iar pentru a o putea folosi va trebui să instalați și *Windows NT 4.0 Service Pack 6*.

În afară de engleză, versiunea *DirectX* 8.0 este localizată în 14 limbi: germană, japoneză, spaniolă, franceză, italiană, portugheză, cehă, poloneză, rusă, suedeză, olandeză, coreeană, chineză simplificată și chineză tradițională. Nu putem decât să sperăm că următoarea versiune va fi localizată și pentru limba română.

DirectX 8.0 este considerat ca fiind o componentă a sistemului de operare și nu poate fi dezinstalat.

Imagine

În versiunile anterioare existau două biblioteci de funcții numite *Direct3D* și *DirectDraw*. În general, nu era nevoie de folosirea simultană a acestora, dar existau și cazuri în care trebuiau apelate funcții din ambele biblioteci. În *DirectX* 8.0 cele două seturi de funcții au fost unificate într-o singură bibliotecă numită *DirectX Graphic*. Prin acest lucru s-a realizat și o reducere semnificativă a complexității inițializării funcțiilor grafice precum și o mai bună utilizare a memoriei. De asemenea au fost incluse funcții care să permită utilizarea noilor facilități oferite de ultimele plăci video apărute pe piață.

Pentru a realiza comunicarea dintre hardware și software a fost creat nivelul HAL (*hardware abstraction layer*), ceea ce ușurează foarte mult și sarcina producătorilor de plăci video. Aceștia pot acum să implementeze diferite funcții la nivelul driver-elor, iar acestea pot fi folosite fără nici o dificultate în aplicații. Un alt nivel creat este HEL (*hardware emulation layer*) care se ocupă cu emularea software a funcțiilor hardware care lipsesc. Mai exact, dacă o placă video nu oferă o anumită funcție, atunci va fi "simulată" existența funcției respective.

Biblioteca *Direct3DX* a fost și ea îmbunătățită. Până acum această bibliotecă era folosită de către *Direct3D* pentru simplificarea tehnicilor folosite de dezvoltatorii 3D. Acum această bibliotecă este folosită de *DirectX Graphic*.

Sunet

Până la versiunea *DirectX* 7.0, efectele sonore erau generate cu ajutorul bibliotecii *DirectX Audio*. Chiar dacă păstrează și acest rol, această componentă are câteva sarcini mai importante cum ar fi aceea de a oferi posibilitatea unor implementări de coloane sonore cu un nivel calitativ ridicat, care să beneficieze de avantajele accelerării hardware. Interfețele *DirectMusic* și *DirectSound* permit programatorilor să insereze în programe secvențe audio fără a mai consuma resurse suplimentare. Formatele suportate acum sunt `.WAV`, `.MIDI` și câteva formate proprii.

Efectele spațiale pot fi și ele mult îmbunătățite generând efecte sonore folosind căi audio diferite pentru a aplica funcțiile de spațializare pentru fiecare sursă de sunet. Până la versiunea *DirectX* 7.0 puteau fi folosite simultan maximum 16 canale MIDI. Începând cu noua versiune, această limitare este înlăturată; acest lucru permite, teoretic, redarea simultană a unui număr oricât de mare de sunete, depășind cu mult performanțele unui sintetizator.

Animație

Arhitectura *DirectShow* permite redarea și capturarea de secvențe multimedia. *DirectShow* beneficiază acum de toate avantajele oferite de *DirectX Graphic* și *DirectSound*. Secvențele video au o calitate mult mai ridicată indiferent dacă se folosește formatul `.MPEG`, `.AVI` sau sunt redate secvențe video stocate pe DVD-uri. În același timp, gradul de ocupare al resurselor rămâne aproximativ același.

Au fost introduse și câteva filtre care permit decompresia fișierelor în format `MPEG` și `MPEG-2`. În ultimul timp DVD-urile sau răspândit foarte mult, mai ales în Statele Unite. Din aceste motive *Microsoft* a decis să ofere câteva noi facilități care să permită programatorului să lucreze mai ușor cu filmele de pe DVD-uri.

Serviciile din categoria *DirectShow Editing* asigură redarea și asamblarea diferitelor secvențe multimedia, precum și realizarea unor interesante efecte audio sau video.

Comunicații

Biblioteca ce se ocupă cu gestionarea conexiunilor în rețea este *DirectPlay*. Aceasta a fost rescrisă pentru a putea beneficia de scalabilitatea crescută a serverelor. Acest lucru permite o creștere semnificativă a vitezei de conectare în rețea.

A apărut *DirectPlay Voice* care își propune să schimbe total conceptul de joc în rețea. Programatorii pot crea acum jocuri în care participanții au la dispoziție tot ceea ce au nevoie pentru a beneficia de comunicare vocală în timp real.

Periferice

DirectInput este biblioteca ce permite comunicarea cu driverele diferitelor periferice instalate. Prin intermediul acesteia, aplicațiile pot primi informații chiar dacă rulează în fundal. Au fost introduse câteva proprietăți noi pentru informațiile provenite de la tastatură, care permit regăsirea rapidă a codurilor tastelor și o combinare mai eficientă a acestora.

Concluzii

DirectX 8.0 aduce o mulțime de îmbunătățiri și noutăți semnificative. Folosind noile facilități, programatorii vor putea să realizeze aplicații mai complexe care să folosească toate avantajele oferite de componentele hardware ale calculatorului pe care sunt rulate.

Chiar dacă majoritatea vechilor aplicații nu pot beneficia de avantajele oferite de *DirectX* 8.0, este recomandată instalarea acestui produs deoarece multe dintre viitoarele aplicații nu vor putea rula corespunzător cu versiuni anterioare.

Unii afirmă că *DirectX* 8.0 va aduce o revoluție în lumea jocurilor pe calculator prin noile facilități pe care le aduce: comunicare vocală în timp real, efecte sonore speciale etc.

Carmen Șușman este redactor al GInfo. Poate fi contactat prin intermediul redacției.

CONCURSUL de programare AGORA

Cea de-a doua rundă a Concursului de Programare Agora s-a încheiat. În acest număr vă vom prezenta soluțiile oficiale ale celor trei probleme, precum și rezultatele obținute de concurenți la această etapă. De asemenea, veți putea vedea care este clasamentul intermediar, după două etape.

Conform așteptărilor noastre, au existat o mulțime de concurenți care nu au reușit să se înscrie în timp util pentru a participa la prima rundă. Din acest motiv, numărul participanților la a doua rundă a fost mai mare decât cel de la prima.

La sfârșitul anului 2000 numărul celor înscriși la *Concursul de Programare Agora* s-a apropiat de 400. Dintre aceștia, mai mult de 100 sunt concurenți din afara României.

Din păcate, au existat câțiva concurenți care, deși au obținut punctaj maxim la prima rundă, nu au mai trimis rezolvări și pentru runda a doua, pierzând astfel șansa de a se afla între primii clasați.

De asemenea, mulți dintre cei care au avut punctaje mari la prima rundă, au avut rezultate aproape dezastruoase la cea de-a doua. Au fost și concurenți cu rezultate mai slabe la prima rundă care au obținut punctaje excelente la cea de a doua.

A doua rundă

Cele trei probleme de la prima rundă nu au mai fost la fel de simple ca cele de la prima. Acest lucru este dovedit de numărul relativ mic al celor care au obținut punctaj maxim. Au acumulat 300 de puncte doar trei concurenți, față de cei 28 de la prima rundă.

Două dintre cele trei probleme necesitau folosirea unui algoritm foarte cunoscut pentru obținerea rezultatului. Cea de-a treia problemă, deși nu era foarte cunoscută, a fost rezolvată de foarte mulți concurenți, deoarece a fost indicată destul de clar o "sursă de inspirație" pentru rezolvare.

La prima rundă trebuiau simulate operații cu numere întregi foarte mari pentru toate problemele. La această rundă, doar una dintre probleme necesita efectuarea de astfel de operații dar, de data aceasta, numerele erau reale.

Erori

Nu am scăpat nici la runda a doua de programele care foloseau unit-ul CRT. Reamintim faptul că evaluarea este realizată pe un calculator dotat cu un procesor *Intel Pentium III* la 500 MHz. Ca urmare, datorită unui bug din *Borland Pascal 7.0*, programele care folosesc unit-ul CRT realizează o împărțire la zero chiar la începutul execuției. Din acest motiv, toate programele care folosesc acest unit nu vor mai furniza rezultatul corect, deoarece vor fi întrerupte înainte de a efectua vreun calcul.

Din nou s-a folosit tipul `Comp` fără a include directiva de compilare `{$N+}`. Din acest motiv câteva programe, care ar fi putut fi corecte, nu au primit nici un punct datorită faptului că nu a fost posibilă compilarea lor.

O categorie destul de ciudată de erori o constituie cele apărute în redactarea fișierelor `RUNDA2.BAT`. Unii au încercat să compileze fișiere `.CPP`, folosind compilatorul pentru *Pascal*. Alții au încercat să compileze fișiere pe care, de fapt, nu le-au trimis, sau le-au trimis cu un alt nume. De asemenea, mulți nici nu au considerat că merită efortul de a mai crea fișierul. La această rundă am creat noi acest fișier, dar începând cu runda a treia nu o vom mai face. Cei care vor trimite arhive care nu conțin acest fișier nu vor primi nici un punct la runda respectivă.

Așteptăm părerile voastre...

Noi am dori ca *CPA* să fie un concurs pe gustul tuturor. Știm că acest lucru nu poate fi realizat, dar cu ajutorul vostru, concursul ar putea fi mai aproape de așteptări. Așadar, nu ezitați să ne scrieți!

RUNDA 2 - Top 50

Mohammad Ali Safari Ghahsareh, Iran 300
Cristian Băicoianu, Ploiești 300
Răzvan Mușaloiu-Elefteri, București 300
Bogdan Dumitru, Ploiești 290
Ciprian Cană, Vaslui 286
Maximilian Machedon, București 274
Cristian Toth, Lugoj 272
George Drumea, București 271
Mihai Stroe, București 263
Andrei Benea, Focșani 254

11. Radu Vătavu, Suceava 238
12. Liviu Lalescu, Craiova 233
13. Claudiu Gruia, București 232
14. Ioana-Maria Ileană, Alba Iulia 224
14. Adrian Cârcu, Bistrița 224
14. Radu Ștefan, Brașov 224
14. Cristian Alexandrescu, Botoșani 224
14. Vlad Vâlceanu, Arad 224
14. Victor Asavei, Dr. Tn. Severin 224
20. Csaba Andras, Oradea 220
20. Csaba Patcas, Oradea 220
22. Gabriel Pârvan, Râmnicu Sărat 218
23. Radu Cocieru, Sibiu 215
24. Daniel Crișan, Râmnicu Vâlcea 209
24. Cosmin Răianu, Constanța 209
26. Denis Bogdănaș, Iași 206
27. Mihai Pătrașcu, Craiova 205
28. Flaviu Pașca, Bistrița 200
28. Vincent Groenhuis, Olanda 200
28. Dan Ghinea, București 200
28. Valentin Bisa, Lugoj 200
28. Mihail Pițu-Rada, Dr. Tn. Severin 200
33. Valentin Pop, Carei 190
34. Paul Marinescu, Buzău 189
34. Cristian Ioniță, București 189
36. Petru-Simon Moț, Brăila 187
37. Daniel Dumitran, București 185
38. Horia Ciurdar, Timișoara 184
39. Mihail Popa, București 172
39. Bogdan Stan, Câmpina 172
41. Adrian Balasko, Zalău 162
42. Andrei David, Râmnicu Vâlcea 161
43. Sorin Oțescu, Brașov 158
44. Dan Popovici, Arad 153
45. Lucian Burja, Blaj 144
46. Vlad Dascălu, Bacău 142
47. Ionuț Andreica, București 139
48. Marius Andrei, București 133
49. Bogdan Piloga, Timișoara 132
49. Lucian-Daniel Stanciu, Brăila 132

Runda 2, problema 1

Acadele

Această problemă a fost rezolvată corect de către 71 dintre cei care au participat la a doua rundă a *Concursului de Programare Agora*. Așa cum se preciza și în enunț rezolvarea putea fi găsită într-o culegere publicată la editura *Computer Libris Agora*. Titlul culegerii este *Un Atelier de programare* și a fost scrisă de *dl. prof. Cristian Giumale*.

Rezolvarea, folosind metoda *backtracking* sau o altă metodă de generare și verificare a tuturor configurațiilor (B,L,D), nu se va încadra în timp deoarece spațiul care trebuie explorat este foarte mare și parametrii problemei depind funcțional unii de alții.

Cele N acadele produse zilnic sunt ambalate câte L în B cutii folosindu-se N ambalaje diferite, corespunzând celor N tipuri disponibile. Rezultă ecuația $B \cdot L = N$.

Restricția, potrivit căreia la sfârșitul fiecărui ciclu de producție, pentru fiecare pereche de ambalaje diferite există o singură cutie care conține exact două acadele cu cele două ambalaje, celelalte având ambalaje diferite față de cele două, poate fi reformulată astfel: *pentru fiecare tip de ambalaj din cele N (fie el x) și pentru fiecare orice alt tip y de ambalaj din cele N - 1 rămase, există o singură cutie cu ambalajele x și y, produsă în cele D zile de producție; într-o cutie, x se află împreună alături de L - 1 ambalaje diferite, iar x nu mai poate apărea în alte cutii produse în aceeași zi. Rezultă că după cele D zile, fiind zilnic alături de L - 1 ambalaje diferite, x trebuie să acopere cele N - 1 posibilități de asociere; se obține ecuația* $D \cdot (L - 1) = (N - 1)$.

La cele două ecuații se adaugă inegalitatea $B > 1$, care este prezentă în enunț.

Se obține următorul sistem nedeterminat:

$$B \cdot L = N \qquad (*)$$
$$D \cdot (L - 1) = (N - 1) \qquad (**)$$
$$B > 1 \qquad (***)$$

În acest sistem valorile L, B, N și D sunt numere întregi pozitive.

În continuare vom arăta că o soluție a acestui sistem este și o soluție a problemei noastre.

Teoremă

Fie L_V o valoare a variabilei L care satisface ecuațiile (*) și (**). Atunci L_V satisface toate restricțiile problemei.

Demonstrație

Fie B_V și D_V valorile întregi corespunzătoare valorii L_V. Deoarece aceste valori sunt obținute din ecuația (*), atunci primele trei restricții din enunț sunt, evident, îndeplinite. Vom arăta acum că și ultima restricție este satisfăcută.

Fie x un ambalaj, ales la întâmplare din cele N. Conform primei restricții, x apare într-o singură cutie produsă într-o anumită zi a ciclului de producție, iar pe întreg ciclul, x apare în D_V cutii. Să presupunem că x nu respectă ultima restricție. Atunci, există cel puțin două cutii care, în afară de x, conțin alte ambalaje comune și, deci, numărul de învelișuri distincte din cele două cutii este mai mic decât $2 \cdot (L_V - 1)$. Înseamnă că cele D_V cutii în care se găsește x conțin împreună $D_V \cdot (L_V - 1) < N - 1$ ambalaje, iar ecuația (**) nu este satisfăcută. Acest lucru este imposibil deoarece ipoteza ne asigură că L_V și D_V satisfac această ecuație. Prin urmare, ultima restricție este satisfăcută pentru x și, deoarece x a fost ales aleator, este satisfăcută pentru oricare dintre cele N ambalaje.

Această teoremă justifică un algoritm de rezolvare foarte simplu: dintre toate soluțiile sistemului ecuațiilor (*), (**), (***) se alege soluția pentru care valoarea $|D - B|$ este minimă.

```
algoritm acadele(N)
  min ← ∞
  pentru L=2,N-1 execută
    dacă rest[N/L]=0 °i rest[(N-1)/(L-1)]=0
    atunci // o soluție posibilă
      B ← N/L
      D ← (N-1)/(L-1)
      dacă |D-B| < min atunci
        Lmin ← L
        Bmin ← B
        Dmin ← D
        min ← |D-B|
      sfâr°it dacă
    sfâr°it dacă
  sfâr°it pentru
    dacă min ≠ ∞ atunci
      returnează Lmin,Bmin,Dmin
    altfel returnează 0,0,0
sfâr°it algoritm
```

Acest algoritm este liniar și funcționează într-un timp acceptabil pentru valori medii ale lui N, dar pentru valori mari nu se va mai încadra în limita de timp admisă.

Din acest motiv algoritmul nu poate rămâne în această formă. Vom prezenta în continuare o adaptare care se va executa într-un timp subliniar și anume $O(\sqrt{N})$.

Problema are două proprietăți importante. Prima restrânge intervalul de căutare a soluției: este suficient ca valorile L care satisfac restricțiile problemei să fie căutate în intervalul $[2, \sqrt{N}]$.

A doua proprietate arată că diferența $D - B$ este pozitivă pentru L cuprins între 2 și $\sqrt{N}$, iar valoarea maximă a diferenței se obține pentru valoarea maximă a lui L, care satisface restricțiile problemei.

Vom demonstra în continuare cele două proprietăți.

Lemă

Nu există nici o valoare $\sqrt{N} < L < N$ pentru care ecuațiile (*) și (**) sunt satisfăcute.

Demonstrație

Fie ecuațiile (*) și (**) rescrise în forma:

```
N mod L = 0                    (*)
(N - 1) mod (L - 1) = 0        (**)
```

Să considerăm că există o valoare $\sqrt{N} < L < N$ care satisface ecuațiile (*) și (**). Atunci, trebuie să existe doi întregi, $q_1 > 0$ și $q_2 > 0$, astfel încât

$$N = q_2 \cdot L$$
$$N - 1 = q_1 \cdot (L - 1)$$

Din aceste ecuații obținem

$$q_1 = \frac{q_2 \cdot L - 1}{L - 1} = \frac{q_2 \cdot (L-1) + q_2 - 1}{L - 1} = q_2 + \frac{q_2 - 1}{L - 1}.$$

Avem $\sqrt{N} < L$; din ecuația $N = q_2 \cdot L$ obținem $q_2 < \sqrt{N}$, deci $q_2 < L$. Atunci, raportul dintre $q_2 - 1$ și $L - 1$ este subunitar și obținem $q_2 \le q_1 < q_2 + 1$.

Deoarece q_1 și q_2 sunt valori întregi rezultă că $q_1 = q_2$. Obținem $N / L = (N - 1) / (L - 1)$, adică

$$N \cdot L - N = N \cdot L - L.$$

Obținem egalitatea $N = L$, ceea ce contravine ipotezei.

Lemă

Valoarea minimă pentru $|D - B|$ se obține pentru cea mai mare valoare L din intervalul $(1, N)$ care satisface ecuațiile (*) și (**).

Demonstrație

Se observă că $D - B$ este o funcție care depinde de L:

$$f(L) = \frac{N-1}{L-1} - \frac{N}{L} = \frac{N-L}{L \cdot (L-1)}.$$

Vom arăta că această funcție este strict descrescătoare pentru valorile L cuprinse în intervalul $[2, \sqrt{N}]$.

Vom considera două valori L_1 și L_2 ($L_1 < L_2$) din acest interval și vom demonstra că $f(L_1) > f(L_2)$.

Trebuie să demonstrăm că

$$\frac{N - L_1}{L_1 \cdot (L_1 - 1)} > \frac{N - L_2}{L_2 \cdot (L_2 - 1)}.$$

Avem $(N - L_1) \cdot L_2 \cdot (L_2 - 1) > (N - L_2) \cdot L_1 \cdot (L_1 - 1)$ care se reduce, după efectuarea unor calcule la

$$(L_2 - L_1) \cdot (N \cdot L_2 + N \cdot L_1 - N - L_1 \cdot L_2) > 0$$

Primul termen al produsului este întotdeauna pozitiv deoarece $L_2 > L_1$.

Al doilea termen poate fi scris sub forma

$$N \cdot (L_2 - 1) + L_1 \cdot (N - L_2).$$

Datorită intervalului ales pentru valorile L_1 și L_2, și acest termen este pozitiv.

În concluzie, inegalitatea inițială este adevărată pentru intervalele care ne interesează, deci funcția $f(L)$ este descrescătoare pe intervalul considerat.

Rezultă imediat că valoarea minimă posibilă a expresiei $|D - B|$ este obținută pentru cea mai mare valoare L din intervalul $(1, N)$ care satisface ecuațiile (*) și (**).

Din aceste două leme și teorema prezentată la început rezultă că soluția problemei este unică și corespunde valorii maxime a parametrului L care satisface ecuațiile (*) și (**).

Se obține astfel următorul algoritm:

```
algoritm acadele2(N)
  pentru L=[sqrt(N)], 2 pas -1 execută
    dacă rest[N/L]=0 °i rest[(N-1)/(L-1)]=0
      atunci
        returnează L, N/L, (N-1)/(L-1)
    sfâr°it dacă
  sfâr°it pentru
  returnează 0, 0, 0
sfâr°it algoritm
```

Acest algoritm poate fi implementat foarte ușor chiar dacă demonstrarea corectitudinii sale a necesitat câteva observații matematice nu foarte ușor de demonstrat.

Vă prezentăm în continuare versiunea oficială a soluției acestei probleme. Aceasta a fost adaptată după rezolvarea propusă de *dl. prof. Cristian Giumale* în culegerea sa, *Un atelier de programare.*

Listing LOL.PAS

```
#include <stdio.h>
#include <math.h>

void main(void){
  long N,L;
  FILE *f=fopen("LOL.IN","rt");
  fscanf(f,"%ld",&N);
  fclose(f);
  for (L=sqrt(N);L>1;L--)
    if (!(N%L) && !((N-1)%(L-1))
      break;
  f=fopen("LOL.OUT","wt");
  if (L==1) fprintf(f,"0 0 0");
  else fprintf("%ld %ld %ld",L,N/L,
                              (N-1)/(L-1));
  fclose(f);
}
```

Au rezolvat corect...

Cristian Alexandrescu, Botoşani
Csaba Andras, Oradea
Marius Andrei, Bucureşti
Mugurel Ionuţ Andreica, Bucureşti
Liviu-Cosmin Andreicuţ, Bucureşti
Victor Asavei, Drobeta Turnu Severin
Constantin Asofiei, Târgu Neamţ
Cristian Băicoianu, Ploieşti
Ciprian-Nicolae Baicu, Dr. Turnu Severin
Adrian Balasko, Zalău
Andrei Benea, Focşani
Valentin Bişa, Lugoj
Denis Bogdănaş, Iaşi
Vencel Bors, Oradea
Lucian Burja, Blaj
Ciprian Cană, Vaslui
Adrian Cârcu, Bistriţa
Horia Ciurdar, Timişoara
Radu Cocieru, Sibiu
Horea Coroiu, Cluj-Napoca
Daniel Crişan, Râmnicu Vâlcea
Andrei Csibi, Cluj-Napoca
Vlad Dascălu, Bacău
Laurent Demonet, Franţa
Andrei-Liviu Dumbrava, Iaşi
Octavian-Daniel Dumitran, Bucureşti
Bogdan Dumitru, Ploieşti
Aurel-Mihai Fulger, Constanţa
Ciprian Gheorghe, Bucureşti
Dan Ghinea, Bucureşti
Andrei Giurgiu, Bucureşti
Mihai Gojol, Cluj-Napoca
Vincent Groenhuis, Olanda
Claudiu Gruia, Bucureşti
Mihai Hărănguş, Cluj-Napoca
Ioana-Maria Ileană, Alba Iulia
Cristian Ioniţă, Bucureşti
Liviu Lalescu, Craiova
Codruţ-Lucian Lazăr, Câmpeni
Maximilian Machedon, Bucureşti
Harsha Madhyastha, India
Paul Marinescu, Buzău
Mihail-Cosmin Piţ-Rada, Dr. Turnu Severin
Alexandru Mosoi, Bacău
Petru-Simon Moţ, Brăila
Raluca Muşaloiu-Elefteri, Bucureşti
Răzvan Muşaloiu-Elefteri, Bucureşti
Venkitasubramaniam Muthuramakrishnan, India
Bogdan Nicolae, Sibiu
Tiberius Pârcălabu, Bucureşti
Flaviu Paşca, Bistriţa
Csaba Patcas, Oradea
Mihai Pătraşcu, Craiova
Rusu Paul, Cluj-Napoca
Liviu Păunescu, Constanţa
Valentin Pop, Carei
Mihail Popa, Bucureşti
Dan Popovici, Arad
Cosmin Răianu, Constanţa
Mohammad Ali Safari Ghahsareh, Iran
Bogdan-Nicolae Şanta, Cluj-Napoca
Lucian-Raul Silistru, Vaslui
Costin Speciac, Buzău
Lucian-Daniel Stanciu, Brăila
Radu Ştefan, Braşov
Radu Ştefan, Curtea de Argeş
Mihai Stroe, Bucureşti
Cristian Toth, Lugoj
Vlad Vâlceanu, Arad
Radu Vătavu, Suceava
Victor Vernescu, Constanţa

Runda 2, problema 2

Puncte

Această problemă a fost rezolvată corect de către 28 dintre cei care au participat la a doua rundă a *Concursului de Programare Agora*. Problema era destul de simplă, un algoritm optim pentru rezolvarea ei putând fi găsit în numeroase cărți care tratează algoritmi care rezolvă probleme de geometrie. Metoda care ar fi trebuit folosită este ***divide et impera***.

Distanța dintre două puncte P_1 de coordonate x_1, y_1 și P_2 de coordonate x_2, y_2 se calculează cu ajutorul formulei

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

Două puncte pot să coincidă, caz în care distanța dintre acestea este 0.

Un algoritm banal care rezolvă problema va verifica distanțele dintre toate perechile de puncte și alegerea minimului acestor distanțe. Acest algoritm are ordinul de complexitate $O(n^2)$ și nu poate fi aplicat în cazul nostru, deoarece putem avea foarte multe puncte în plan (maximum 10000).

Pentru a obține un algoritm cu o complexitate mai bună ($O(n \cdot \log n)$) vom folosi metoda ***divide et impera***.

Un apel recursiv al algoritmului va avea ca intrare o submulțime P a mulțimii punctelor și două șiruri X și Y. Punctele din șirul X sunt ordonate în așa fel încât coordonatele x să fie în ordine crescătoare. De asemenea, punctele din șirul Y sunt ordonate în așa fel încât coordonatele y să fie crescătoare.

Pentru a obține ordinul de complexitate dorit, nu ne permitem să efectuăm sortări la fiecare apel recursiv. Pentru a evita acest lucru vom folosi o metodă numită ***presortare***, cu ajutorul căreia vom menține șirurile ordonate fără a le sorta efectiv la fiecare apel recursiv.

Un apel recursiv cu intrările P, X și Y va verifica mai întâi dacă în mulțimea P există mai mult de trei puncte. Dacă avem cel mult trei puncte, atunci vom verifica distanțele dintre toate perechile de puncte (avem o pereche dacă sunt două puncte și trei perechi dacă sunt trei puncte; nu este posibil să avem un punct) și o vom alege pe cea minimă. Pentru mulțimi cu mai mult de trei puncte vom aplica cele trei etape ale algoritmilor divide et impera: *divide*, *stăpânește* și *combină*.

Etapa divide

Vom determina o dreaptă verticală d care împarte mulțimea punctelor P în două submulțimi P_S și P_D, fiecare conținând jumătate din punctele mulțimii P. Dacă numărul de puncte din P este impar, atunci una dintre mulțimi va conține cu un punct mai mult decât cealaltă. Toate punctele din P_S vor fi pe dreapta d sau în stânga acesteia și toate punctele din P_D sunt pe dreapta d sau în dreapta acesteia.

Șirul X este împărțit în șirurile X_S și X_D care conțin punctele din P_S, respectiv P_D, ordonate crescător după coordonata x. Analog, șirul Y este împărțit în șirurile Y_S și Y_D care conțin punctele din P_S, respectiv P_D, ordonate crescător după coordonata y.

Etapa stăpânește

Având împărțită mulțimea P în submulțimile P_S și P_D, vom efectua două apeluri recursive, unul pentru a găsi cea mai apropiată pereche de puncte din P_S și celălalt pentru a găsi cea mai apropiată pereche din P_D.

Pentru primul apel vom folosi ca intrări mulțimea P_S și șirurile X_S și Y_S, iar pentru al doilea mulțimea P_D și șirurile X_D și Y_D. Fie d_S și d_D valorile returnate în urma celor două apeluri și D minimul acestora.

Etapa combină

Cea mai apropiată pereche de puncte din mulțimea P este, fie cea aflată la distanța D, fie o altă pereche de puncte, unul aflat în mulțimea P_S și celălalt aflat în mulțimea P_D. Va trebui să verificăm dacă există o astfel de pereche aflată la o distanță mai mică decât D. Observăm că, dacă există o astfel de pereche, atunci aceste puncte se află la o distanță de cel mult D față de dreapta d. Ca urmare, ele trebuie să se afle într-o regiune de lățime $2 \cdot D$ centrată în jurul dreptei d. Pentru a găsi o pereche vom proceda astfel:

- Construim un șir Y' care conține toate punctele din șirul Y, care se află în interiorul regiunii de lățime $2 \cdot D$. Acest șir va fi sortat crescător după coordonata y a punctelor.
- Pentru fiecare punct p din șirul Y' vom încerca să găsim punctele din Y' care se află la o distanță cel mult D față de p. Așa cum vom arăta ulterior, este necesar să fie considerate doar șapte puncte din Y', care urmează după p. Vom calcula distanțele de la p la cele șapte puncte și vom reține distanța minimă D' a perechii celei mai apropiate.
- Dacă $D' < D$ atunci regiunea verticală conține o pereche mai apropiată decât cea găsită prin apelurile recursive. În acest caz este returnată această pereche și distanța sa D'. Altfel, este returnată perechea cea mai apropiată găsită prin apelurile recursive, împreună cu distanța sa D.

În această descriere a algoritmului am omis câteva detalii de implementare care sunt necesare pentru a obține ordinul de complexitate $O(n \cdot \log n)$. Vom reveni asupra acestora după ce vom demonstra corectitudinea algoritmului.

Corectitudinea acestui algoritm este evidentă, cu excepția a două aspecte. Primul constă în oprirea apelurilor recursive atunci când avem mai puțin de patru puncte deoarece trebuie să ne asigurăm că nu vom încerca niciodată să împărțim o mulțime care conține un singur punct. Al doilea aspect constă în demonstrarea faptului că avem nevoie de verificarea a doar șapte puncte pentru fiecare punct p din Y'. Această proprietate este demonstrată în continuare.

Să presupunem că la un anumit nivel al recursivității, punctele cele mai apropiate se află unul în mulțimea P_S și unul în mulțimea P_D. Fie acestea p_S și p_D; distanța D' dintre acestea este strict mai mică decât D. Punctul p_S trebuie să fie pe dreapta d sau în stânga ei, iar punctul p_D trebuie să fie pe dreapta d sau în dreapta ei; cele două puncte trebuie să se afle la o distanță de cel mult D unități față de dreapta d. Mai mult, distanța pe verticală dintre p_S și p_D trebuie să fie cel mult D. În concluzie, p_S și p_D se află într-un dreptunghi de lungime $2 \cdot D$ și înălțime D, centrat pe dreapta d.

Vom arăta în continuare că există cel mult opt puncte care se pot afla în interiorul unui astfel de dreptunghi. Dreapta d împarte dreptunghiul în două pătrate cu latura de lungime D.

Să considerăm acum pătratul din stânga dreptei d. Deoarece toate punctele din P_S se află la o distanță de cel puțin D față de p_S (altfel după apelul recursiv ar fi fost returnată o altă distanță mai mică), cel mult patru puncte pot fi situate în interiorul acestui pătrat.

În mod analog putem demonstra că cel mult patru puncte se pot situa în pătratul din dreapta dreptei d.

În concluzie, cel mult opt puncte din P se pot situa în interiorul dreptunghiului. Din acest motiv, este suficient să verificăm cel mult șapte puncte care urmează după fiecare punct din Y'.

Presupunem, fără a reduce generalitatea, că punctul p_S se află înaintea punctului p_D în șirul Y'. În acest caz, chiar dacă p_S apare cât mai devreme în șirul Y' și p_D apare cât mai târziu în acest șir, p_D se află pe una dintre cele șapte poziții care urmează după p_S.

Cu aceasta am demonstrat corectitudinea algoritmului de determinare a celei mai apropiate perechi de puncte.

Pentru a obține complexitatea dorită trebuie să ne asigurăm că punctele din șirurile X_S, X_D, Y_S și Y_D care sunt folosite în apelurile recursive, sunt sortate după coordonatele x, respectiv y. De asemenea, punctele din șirul Y' trebuie să fie sortate după coordonate y. Observăm că dacă șirul punctelor din mulțimea P este deja sortat, împărțirea mulțimii P în submulțimile P_S și P_D se realizează foarte ușor în timp liniar.

Observația cheie a rezolvării acestei probleme este aceea că dorim să obținem un subșir sortat al unui șir sortat. Fie un apel particular, având ca date de intrare mulțimea P și șirul Y sortate după coordonata y. După ce am împărțit mulțimea P în submulțimile P_S și P_D, trebuie să formăm în timp liniar șirurile Y_S și Y_D, care vor fi sortate după coordonata y. Pentru aceasta vom examina, în ordine, punctele din șirul Y. Dacă un anumit punct se află în mulțimea P_S atunci îl adăugăm la sfârșitul șirului Y_S, iar dacă se află în mulțimea P_D atunci îl adăugăm la sfârșitul șirului Y_D. Șirurile X_S, X_D și Y' sunt create în mod analog.

Mai rămâne de văzut cum sortăm punctele la început. Vom face o simplă ***presortare*** a lor, adică le vom sorta, o dată, înainte de primul apel recursiv. Șirurile sortate sunt transmise ca parametri în primul apel și apoi sunt reduse pe măsură ce se avansează în recursivitate. Presortarea poate fi realizată într-un timp $O(n \cdot \log n)$.

Vom analiza acum complexitatea algoritmului ***divide et impera***. Să notăm cu $T(n)$ ordinul de complexitate al acestuia. La fiecare pas vom avea două apeluri recursive ale aceluiași algoritm pentru jumătate din puncte la care se adaugă câteva operații care pot fi efectuate în timp liniar. Putem scrie relația de recurență pentru ordinul de complexitate al algoritmului ca fiind $T(n) = 2 \cdot T(n / 2) + O(n)$. Această formulă este valabilă pentru $n > 3$ deoarece ne oprim din recursivitate dacă nu avem cel puțin patru puncte. Pentru $n < 3$ ordinul este $O(1)$, deoarece sunt efectuate un număr constant de operații.

Din formula recursivă a ordinului de complexitate rezultă că acesta este $O(n \cdot \log n)$. Mai trebuie adăugat și algoritmul de presortare, dar ordinul de complexitate nu se schimbă deoarece există metode de sortare care funcționează într-un timp $O(n \cdot \log n)$.

Aceasta a fost o prezentare generală a algoritmului. Ea a fost preluată din cartea *Introducere în algoritmi* scrisă de *T.H. Cormen, C.E. Leiserson, R.R. Rivest*. Evident, în implementare sunt folosite anumite "trucuri" care reduc timpul de execuție. Din acest motiv, versiunea oficială a rezolvării acestei probleme nu respectă toți pașii care au fost descriși aici.

Listing POINTS.CPP

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <values.h>

float huge x[10000], huge y[10000];
int huge px[10000], huge py[20000];
float min,x1,y1,x2,y2;
int n;

void ReadData(void){
  FILE *f=fopen("POINTS.IN","rt");
  fscanf(f,"%d",&n);
  for (int i=0;i<n;i++)
    fscanf(f,"%f%f",&x[i],&y[i]);
}
```

```
void SortX(int left,int right){
  int i=left,
      j=right,
      k=x[px[(left+right)/2]];
  float aux;
  do{
    while (x[px[i]]<k)
      i++;
    while (x[px[j]]>k)
      j--;
    if (i<=j){
      aux=px[i];
      px[i]=px[j];
      px[j]=aux;
      i++; j--;
    }
  } while (i<=j);
  if (j>left)
    SortX(left,j);
  if (i<right)
    SortX(i,right);
}

void SortY(int left,int right){
  int i=left,
      j=right,
      k=y[py[(left+right)/2]];
  float aux;
  do{
    while (y[py[i]]<k)
      i++;
    while (y[py[j]]>k)
      j--;
    if (i<=j){
      aux=py[i];
      py[i]=py[j];
      py[j]=aux;
      i++; j--;
    }
  } while (i>=j);
  if (j>left)
    SortY(left,j);
  if (i<right)
    SortY(i,right);
}

void SortData(void){
  for (int i=0;i<n;i++)
    px[i]=py[i]=i;
  SortX(0,n-1);
  SortY(0,n-1);
  for (i=0;i<n-1;i++)
    if (x[px[i]]==x[px[i+1]] &&
                    y[px[i]]==y[px[i+1]]){
      FILE *f=fopen("POINTS.OUT","wt");
      fprintf(f,"%.3f %.3f\n%.3f %.3f\n",
       x[px[i]],y[px[i]],x[px[i]],y[px[i]]);
      fclose(f); exit(0);
    }
}

inline float dist(int i,int j){
  return sqrt((x[py[i]]-x[py[j]])*
              (x[py[i]]-x[py[j]])+
              (y[py[i]]-y[py[j]])*
              (y[py[i]]-y[py[j]]));
}

void DivideEtImpera
          (int i1,int j1,int i2,int j2){
  int i,j; float half;
  if (j2-i2<3){
    for (i=i2;i<=j2-1;i++)
      for (j=i+1;j<=j2;j++)
        if (dist(i,j)<min){
          min=dist(i,j);
          x1=x[py[i]];
          y1=y[py[i]];
          x2=x[py[j]];
          y2=y[py[j]];
        }
    return;
  }
  half=x[px[(i1+j1)/2]];
  j=0;
  for (i=i2;i<=j2;i++)
    if (x[py[i]]<half)
      py[j2+(++j)]=py[i];
  if (j>1)
    DivideEtImpera(i1,i1+j-1,j2+1,j2+j);
  j=0;
  for (i=i2;i<=j2;i++)
    if (x[py[i]]>half)
      py[j2+(++j)]=py[i];
  if (j>1)
    DivideEtImpera(j1-j+1,j1,j2+1,j2+j);
  j=0;
  for (i=i2;i<=j2;i++)
    if (fabs(x[py[i]]-half)<min)
      py[j2+(++j)]=py[i];
  for (i=j2+1;i<j2+j;i++)
    for (int m=i+1;m<i+8 && m<=j2+j;m++)
      if (dist(i,m)<min){
        min=dist(i,m);
        x1=x[py[i]];
        y1=y[py[i]];
        x2=x[py[m]];
        y2=y[py[m]];
      }
}
```

```
void Solve(void){
  min=MAXFLOAT;
  DivideEtImpera(0,n-1,0,n-1);
}

void WriteSolution(void){
  FILE *f=fopen("POINTS.OUT","wt");
  fprintf(f,"%.3f %.3f\n%.3f %.3f\n",
                              x1,y1,x2,y2);
  fclose(f);
}

void main(void){
  ReadData();
  SortData();
  Solve();
  WriteSolution();
}
```


Au rezolvat corect...

Cristian Alexandrescu, Botoșani
Csaba Andras, Oradea
Victor Asavei, Drobeta Turnu Severin
Cristian Băicoianu, Ploiești
Valentin Bișa, Lugoj
Ciprian Cană, Vaslui
Adrian Cârcu, Bistrița
Radu Cocieru, Sibiu
Daniel Crișan, Râmnicu Vâlcea
Andrei David, Râmnicu Vâlcea
Vincent Groenhuis, Olanda
Claudiu Gruia, București
Ioana-Maria Ileană, Alba Iulia
Artur Kuczapski, Oradea
Maximilian Machedon, București
Mihail-Cosmin Pițt-Rada, Dr. Turnu Severin
Răzvan Mușaloiu-Elefteri, București
Sorin Oțescu, Brașov
Gabriel Pârvan, Râmnicu Sărat
Flaviu Pașca, Bistrița
Csaba Patcas, Oradea
Cosmin Răianu, Constanța
Mohammad Ali Safari Ghahsareh, Iran
Bogdan Stan, Câmpina
Mihai Stroe, București
Radu Ștefan, Brașov
Cristian Toth, Lugoj
Vlad Vâlceanu, Arad

2001: O odisee spațială...

Un monolit misterios a apărut într-un parc din Seattle pentru a ne reaminti de structura similară din filmul sau cartea "2001: O odisee spațială".

La fel ca și în cazul monolitului din filmul sau romanul SF, se pare că nimeni nu știe cine a construit această structură bizară. Dimensiunile blocului din Seattle respectă raportul 1/4/9 care apare și în "2001: O odisee spațială".

Există puține indicii referitoare la originea construcției din Seattle. Pe ea nu a fost amplasată nici o plăcuță comemorativă, iar pământul a fost aranjat după ridicarea structurii. Totuși, prezența unor capace de plastic în jurul construcției demonstrează că cei care au amplasat-o sunt niște petrecăreți pământeni și nu reprezentanți ai unei civilizații extraterestre.

Chiar dacă se spune că "2001: O odisee spațială" este filmul preferat al lui Bill Gates, nimeni nu presupune că cei de la Microsoft au construit monolitul. Se spune că ideea a fost una mult prea inteligentă pentru a fi a celor din Redmond.

Principalii suspecți sunt membrii unui grup numit Support the Monolith care, pe 31 decembrie 2000, au efectuat o paradă cu un monolit de lemn pe străzile din Seattle.

Poliția a confiscat monolitul de lemn cam în aceeași perioadă în care a fost ridicat cel de oțel.

Runda 2, problema 3

Determinant

Această problemă a fost rezolvată corect doar de 7 dintre cei care au participat la a doua rundă a *Concursului de Programare Agora*. Numai 85 de concurenți au reușit să obțină puncte la această problemă, majoritatea însă au acumulat mai puțin de 10 puncte.

În principiu, problema nu era foarte dificilă, algoritmi pentru rezolvarea sa putând fi găsiți în numeroase cărți de specialitate. Se pare că cei mai mulți au fost descurajați atunci când au văzut că trebuie să implementeze operații cu numere reale foarte mari. Puțini s-au hotărât să încerce acest lucru, iar unii, mai inventivi, au descărcat de pe *Internet* clase C++ care implementau aceste operații.

După rezolvarea problemei referitoare la numerele mari nu mai rămânea decât să se găsească un algoritm cât mai eficient pentru calcularea determinantului unei matrice. Se poate demonstra matematic că un algoritm care rezolvă această problemă are o complexitate minimă de $O(n^{2+\varepsilon})$ unde $\varepsilon \to 0$, dar nu ajunge niciodată la 0. În acest moment, cel mai performant algoritm pentru calcularea determinantului unei matrice are un ordin de complexitate de aproximativ $O(n^{2.3})$. Totuși, dimensiunea datelor de intrare permitea folosirea unui algoritm având complexitatea $O(n^3)$.

O metodă de a calcula determinantul unei matrice a fost prezentată în cadrul articolului *LU*-factorizare din numărul 7/2000 al *GInfo*. În acel articol era prezentat modul în care poate fi descompusă o matrice A într-un produs de matrice $L \cdot U$, dintre care prima era inferior triunghiulară (avea doar valori 0 deasupra diagonalei principale), iar una era superior triunghiulară (avea doar valori 0 sub diagonala principală). Mai mult, elementele de pe diagonala principală a matricei U aveau toate valoarea 1.

Folosind formula $\det A \cdot B = \det A \cdot \det B$ rezultă că $\det A = \det L \cdot \det U$. Deoarece matricele L și U sunt triunghiulare, valorile determinanților lor pot fi obținute înmulțind elementele de pe diagonalele principale. Determinantul matricei U va fi 1, deci determinantul matricei A va fi egal cu determinantul matricei L.

În principiu, tot ce avem de făcut este să calculăm o *LU*-factorizare a matricei date și să înmulțim valorile de pe diagonala principală a matricei L. Apare un singur inconvenient; teorema care stătea la baza algoritmului de *LU*-factorizare prevedea că matricele obținute prin tăierea ultimelor k linii și a ultimelor k coloane trebuie să aibă determinant nenul.

Au rezolvat corect:

Cristian Băicoianu, Ploiești
George Drumea, București
Bogdan Dumitru, Ploiești
Dan Ghinea, București
Aurelian Ghiță, Buzău
Răzvan Mușăloiu-Elefteri, București
Mohammad Ali Safari Ghahsareh, Iran

Chiar dacă este puțin probabil ca acest caz să apară, el nu poate fi ignorat. Din aceste motive, trebuie găsit un algoritm care să rezolve și această problemă, aparent minoră.

Soluția o constituie algoritmul de *LUP*-factorizare care, dându-se o matrice A, determină trei matrice L, U și P astfel încât $P \cdot A = L \cdot U$.

Matricele L și U au aceeași semnificație ca și în cazul *LU*-factorizării, iar matricea P este o matrice de permutare, adică o matrice care conține doar elemente 0 și 1, și suma elementelor de pe fiecare linie și de pe fiecare coloană este 1.

Cu alte cuvinte, fiecare linie și fiecare coloană a matricei P conține exact un 1, restul elementelor fiind 0.

Cazul în care algoritmul de *LU*-factorizare nu funcționa corect era atunci când pe diagonala matricei L se obținea valoarea 0. Apariția matricei P ne asigură acum că dacă pe diagonala matricei L apare valoarea 0, atunci valoarea determinantului este 0.

Este evident faptul că determinantul matricei P este 1, deci vom avea și de această dată $\det A = \det L$.

Elementele matematice aflate în spatele *LUP*-factorizării sunt aceleași ca și în cazul *LU*-factorizării. Algoritmul poate fi adaptat destul de ușor.

```
algoritm determinant(A)
  det ← 1
  n ← numărul liniilor matricei A
  pentru i ← 1,n execută                (*)
    p[i] ← i                            (*)
  sfârșit pentru
```

```
pentru k ← 1,n executã
  pivot ← 0
  pentru i ← k,n executã
    dacã |a_ik|>pivot
      atunci
        pivot ← |a_ik|
        k' ← i
    sfârºit dacã
  sfârºit pentru
  dacã pivot=0 atunci
    returneazã 0
  sfârºit dacã
  interschimbã p[k] cu p[k']          (*)
  pentru i ← 1,n executã
    interschimbã a_ki cu a_k'i
  sfârºit pentru
  pentru i ← k+1,n executã
    a_ik ← a_ik / a_kk
    pentru j ← k+1,n executã
      a_ij ← a_ij - a_ik · a_kj
    sfârºit pentru
  sfârºit pentru
  det ← det · a_kk
sfârºit pentru
returneazã det
sfârºit algoritm
```

Se observă că, la fel ca și în cazul *LU*-factorizării, matricea *A* este factorizată "pe loc". Matricea *P* este păstrată în vectorul *p*; p_i este coloana pe care se află elementul de pe linia *i*, care are valoarea 1.

Pentru a calcula determinantul nu mai este nevoie să determinăm și matricea *P*, motiv pentru care liniile marcate cu `(*)` pot fi eliminate.

Din motive de spațiu nu putem prezenta sursa completă pentru varianta oficială de rezolvare a acestei probleme. Am "eliminat" liniile de cod în care erau implementate operațiile cu numere mari. Vom presupune că acestea sunt definite într-o clasă numită `BigReal` care conține o metodă pentru inițializarea cu un număr real, una pentru înmulțirea cu un număr real și una care returnează partea zecimală a numărului sub forma unui șir de caractere. Definirea acestei clase ar fi trebuit să apară în loc de *#include <bigreal.h>*. Vă prezentăm în continuare această variantă.

Listing DET.CPP

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <bigreal.h>

long double huge a[200][200];
int n;
BigReal det;

void ReadData(void){
  FILE *f=fopen("DET.IN","rt");
  fscanf(f,"%d",&n);
  for (int i=0;i<n;i++)
    for (int j=0;j<n;j++)
      fscanf(f,"%Lf",&a[i][j]);
  fclose(f);
}

void Zero(void){
  FILE *f=fopen("DET.OUT","wt");
  fprintf(f,"0");
  fclose(f);
  exit(0);
}

void LUP(void){
  int kp;
  long double aux;
  det.init(1);
  for (int k=0;k<n;k++){
    long double p=0;
    for (int i=k;i<n;i++)
      if (fabsl(a[i][k])>p){
        p=fabsl(a[i][k]);
        kp=i;
      }
    if (!p)
      Zero();
    for (i=k;i<n;i++){
      aux=a[k][i];
      a[k][i]=a[kp][i];
      a[kp][i]=aux;
    }
    for (i=k+1;i<n;i++){
      a[i][k]/=a[k][k];
      for (int j=k+1;j<n;j++)
        a[i][j]-=a[i][k]*a[k][j];
    }
    det.multiply(a[k][k]);
  }
}

void WriteSolution(void){
  FILE *f=fopen("DET.OUT","wt");
  fprintf(f,"%s",
          det.TheDecimalPartAsAString());
  fclose(f);
}

void main(void){
  ReadData();
  LUP();
  WriteSolution();
}
```

Termen: 25 Februarie 2001!

Runda 4

Vă prezentăm în continuare cele trei probleme de la cea de-a patra rundă a Concursului de Programare Agora. Concurenții care nu s-au înscris până la data de 10 Ianuarie 2000 pot participa la concurs începând cu această rundă, dacă trimit mesajul de înscriere până pe data de 20 Februarie 2001.

CPAR4P1: Sfere

Nume fișier sursă: `SPHERE.PAS`, `SPHERE.C` sau `SPHERE.CPP`
Nume fișier de intrare: `SPHERE.IN`
Nume fișier de ieșire: `SPHERE.OUT`
Descriere: Dacă avem o sferă atunci spațiul este împărțit în două zone: cea din interiorul sferei și cea din exterior. Dacă avem două sfere atunci spațiul poate fi împărțit în trei sau patru zone. Dacă sferele nu se intersectează vom avea trei zone: interiorul primei sfere, interiorul celei de-a doua sfere și zona exterioară sferelor. Dacă sferele se intersectează atunci vor fi patru zone: zona comună celor două sfere, zona din interiorul primei sfere (fără zona comună), zona din interiorul celei de-a doua sfere (fără zona comună) și zona din exteriorul sferelor. Dându-se n ($2 \le n \le 1000$) sfere, determinați care este numărul maxim de zone în care poate fi împărțit spațiul aranjând sferele într-un mod convenabil.
Date de intrare: Fișierul de intrare va conține pe prima linie numărul n al sferelor.
Date de ieșire: Fișierul de ieșire va conține numărul maxim de zone în care poate fi împărțit spațiul.
Exemplu:

`SPHERE.IN`	`SPHERE.OUT`
2	4

Timp de execuție: 1 secundă/test.

CPAR4P2: Asemănare

Nume fișier sursă: `POLY.PAS`, `POLY.C` sau `POLY.CPP`
Nume fișier de intrare: `POLY.IN`
Nume fișier de ieșire: `POLY.OUT`
Descriere: Se dau două poligoane în plan, fiecare având n ($3 \le n \le 100$) vârfuri. Poligoanele sunt date prin coordonatele vârfurilor lor (numere reale cu o precizie de trei zecimale, din intervalul [0, 1000]), în ordine trigonometrică, sau invers trigonometrică. Se cere să se verifice dacă cele două poligoane sunt *asemenea.*
Date de intrare: Prima linie a fișierului de intrare conține numărul n al vârfurilor poligoanelor. Următoarele n linii conțin coordonatele vârfurilor primului poligon. Următoarele n linii conțin coordonatele vârfurilor celui de-al doilea poligon.
Date de ieșire: Dacă poligoanele sunt asemenea, fișierul de ieșire va conține n numere, pe câte o linie, a i-a linie indicând vârful din al doilea poligon care corespunde celui de-al i-lea vârf din primul poligon (unghiurile corespunzătoare acestor două vârfuri trebuie să fie egale). Dacă poligoanele nu sunt asemenea, fișierul de ieșire va conține doar cifra 0.
Exemplu:

`POLY.IN`	`POLY.OUT`
5	2
0.000 0.000	1
0.000 4.000	5
2.000 6.000	4
4.000 4.000	3
4.000 0.000	
0.000 1.000	
0.000 3.000	
2.000 3.000	
2.000 1.000	
1.000 0.000	

Timp de execuție: 1 secundă/test.

CPAR4P3: Dreptunghi

Nume fișier sursă: `RECT.PAS`, `RECT.C` sau `RECT.CPP`
Nume fișier de intrare: `RECT.IN`
Nume fișier de ieșire: `RECT.OUT`
Descriere: Se consideră un dreptunghi ale cărui dimensiuni sunt numere naturale cuprinse între 1 și 100. Dreptunghiul trebuie descompus într-un număr minim de pătrate ale căror laturi sunt paralele cu laturile dreptunghiului. Într-un dreptunghi poate fi efectuată o tăietură completă (de la o margine la alta) paralelă cu o latură.
Date de intrare: Prima linie a fișierului de intrare conține valorile m și n, separate printr-un spațiu, reprezentând dimensiunile dreptunghiului.
Date de ieșire: Fișierul de ieșire va conține numărul minim de pătrate în care poate fi împărțit dreptunghiul, respectând regula de efectuare a tăieturilor.
Exemplu:

`RECT.IN`	`RECT.OUT`
2 4	2

Timp de execuție: 1 secundă/test.

Ginfo Top 100

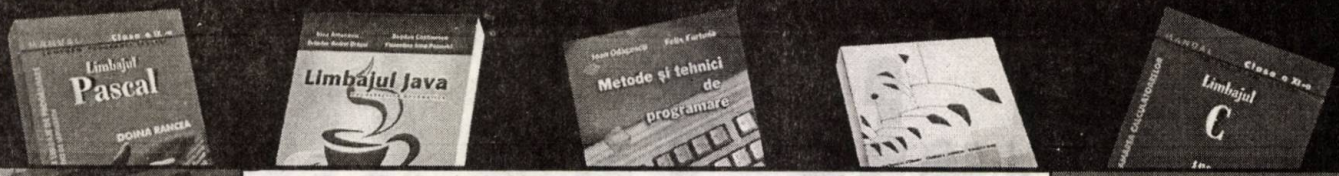
1. Mohammad Ali Safari Ghahsareh, Iran 600

2. Bogdan Dumitru, Ploiești 590

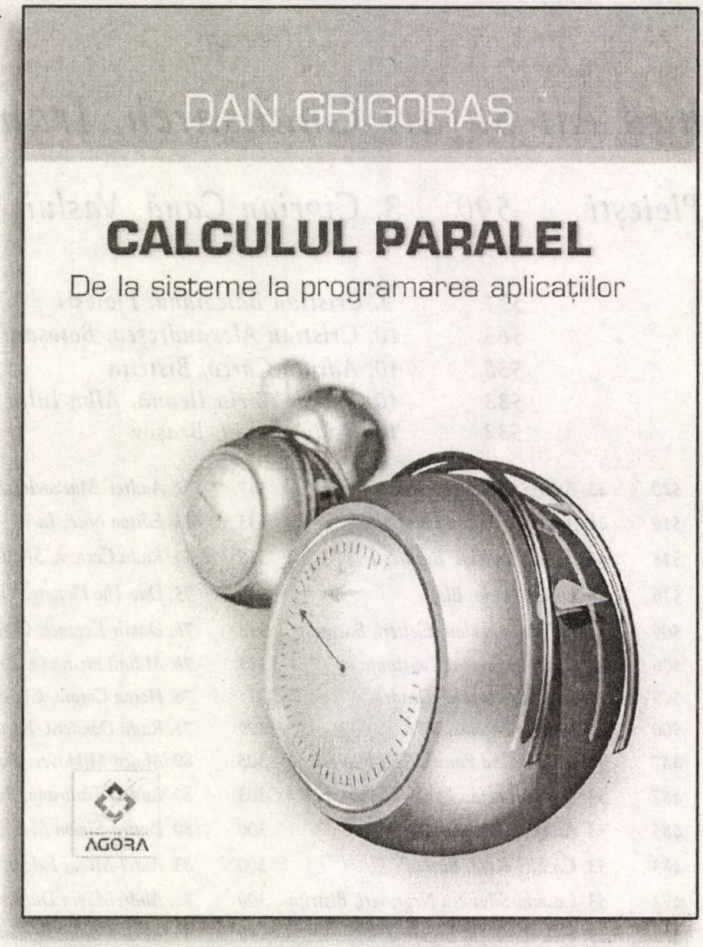
3. Ciprian Cană, Vaslui 586

4. George Drumea, București 571
5. Mihai Stroe, București 563
6. Radu, Vătavu, Suceava 538
7. Liviu Lalescu, Craiova 533
8. Claudiu Gruia, București 532
9. Cristian Băicoianu, Ploiești 530
10. Cristian Alexandrescu, Botoșani 524
10. Adrian Cârcu, Bistrița 524
10. Ioana-Maria Ileană, Alba Iulia 524
10. Radu Ștefan, Brașov 524

14. Csaba Andras, Oradea 520
15. Vlad Vâlceanu, Arad 519
16. Gabriel Pârvan, Râmnicu Sărat 518
17. Răzvan Mușaloiu-Elefteri, București 516
18. Daniel Crișan, Râmnicu Vâlcea 509
19. Denis Bogdănaș, Iași 506
20. Mihai Pătrașcu, Craiova 505
21. Flaviu Pașca, Bistrița 500
22. Vincent Groenhuis, Olanda 487
22. Dan Ghinea, București 487
24. Daniel Dumitran, București 485
25. Cosmin Răianu, Constanța 483
26. Mihail Popa, București 472
27. Bogdan Stan, Câmpina 472
28. Horia Ciurdar, Timișoara 467
29. Sorin Oțescu, Brașov 458
30. Mugurel Ionuț Andreica, București 432
31. Vlad Dascălu Bacău, 429
32. Dan Popovici Arad, 428
33. Victor Asavei, Dr. Turnu Severin 424
34. V. Muthuramakrishnan, India 421
35. Liviu Păunescu, Constanța 416
36. Vencel Bors, Oradea 409
37. Ciprian Baicu, Dr. Turnu Severin 406
38. Aurelian Ghiță, Buzău 388
39. Alexandra Constantin, Ploiești 374
40. Csaba Patcas, Oradea 366
41. Paul Marinescu, Buzău 365
42. Cristian Toth, Lugoj 352
43. Andrei Csibi, Cluj-Napoca 347
44. Andrei David, Râmnicu Vâlcea 333
45. Raluca Sauciuc, București 330
46. Lucian Burja, Blaj 320
47. Raluca Mușaloiu-Elefteri, București 316
48. Victor Vernescu, Constanța 315
49. Artur Kuczapski, Oradea 311
50. Bogdan Nicolae, Sibiu 309
51. Mihai-Vlad Pantiș, Cluj-Napoca 306
52. Codruț-Lucian Lazăr, Câmpeni 303
53. Arcadie Crăcan, Iași 300
53. Cătălin Kesa, Bușteni 300
53. Cosmin-Silvestru Negruseri, Bistrița 300
56. Valentin Bisa, Lugoj 299
57. Mihai Gojol, Cluj-Napoca 296
58. Maximilian Machedon, București 274
59. Mihai Hărănguș, Cluj-Napoca 271
60. Preslav Nakov, Bulgaria 266
61. Narayanan Arvind, India 264
62. Andrei Benea, Focșani 254
63. Bogdan-Milovan Piloga, Timișoara 254
63. Andrei Giurgiu, București 254
65. Bogdan Șanta, Cluj-Napoca 251
66. Adrian Cearnău, București 250
67. Rusu Paul, Cluj-Napoca 246
68. Cristian Ioniță, București 243
69. Petko Minkov, Bulgaria 241
70. Valentin Pop, Carei 235
71. Andrei Adler, București 228
72. Andrei Markovits, Satu Mare 223
73. Edison Nica, Iași 216
74. Radu Cocieru, Sibiu 215
75. Dao The Phuong, Vietnam 203
76. Damir Korencic Croația 200
76. Mihail Pițu-Rada, Dr. Turnu Severin 200
78. Horea Coroiu, Cluj-Napoca 196
78. Radu Dondera, București 196
80. Mojca Miklavec, Slovenia 187
80. Andrei Tătăranu, Buzău 187
80. Pentru-Simon Moț, Brăila 187
83. Aurel-Mihai Fulger, Constanța 181
84. Andrei-Liviu Dumbrava, Iași 164
85. Adrian Balasko, Zalău 162
86. Constantin Asofiei, Târgu Neamț 158
87. Harsha Madhyastha, India 153
88. Radu Ștefan, Curtea de Argeș 140
89. Liviu-Cosmin Andreicuț, București 139
90. Alina Fești, Cluj-Napoca 135
91. Marius Andrei, București 133
92. Lucian-Daniel Stanciu, Brăila 132
93. Cristian Bojinovici, București 129
94. Silviu-Gabriel Udrea, Târgu Jiu 129
95. Nadina Busuioc, Constanța 124
96. Ciprian Gheorghe, București 120
97. Daniel Comșa, Crăciunelu de Jos 118
98. Ștefan Bucur, București 117
99. Costin Speciac, Buzău 116
100. Ionel-Cristian Iană, București 115



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate, dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performață și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

AGORA

Configurații ORDONATE

Mihai Oltean, Crina Groșan

Exsită numeroase probleme în care se dau anumite configurații între care poate fi stabilită o relație de ordine. În cadrul acestui articol vă vom prezenta câteva exemple în care trebuie să determinăm numărul de ordine al unei configurații sau configurația care are un anumit număr de ordine.

Exemplele care urmează au toate același enunț general: *se consideră o mulțime de configurații înzestrate cu o relație de ordine totală (oricare două configurații - elemente ale mulțimii - pot fi comparate).*

Fie K cardinalul mulțimii de configurații. Din motivul mai sus enunțat se poate stabili o relație bijectivă între mulțimea configurațiilor și mulțimea $\{1,2,\ldots,K\}$.

Numărul i asociat unei configurații se numește număr de ordine al configurației respective în cadrul mulțimii de configurații.

Apar două probleme:

- Se dă o configurație și se cere numărul ei de ordine.
- Se dă un număr de ordine și se cere configurația atașată.

Nume din cartea de telefon

Se consideră numele persoanelor din cartea de telefon. Pentru simplificare se presupune că numele sunt formate dintr-un singur cuvânt de lungime m, care conține litere doar din primele n litere ale alfabetului.

Observație

Toate numele, care pot fi formate cu literele date, există în cartea de telefon. Nu există două persoane cu același nume.

Se dă un nume de persoană și se cere numărul lui de ordine din cartea de telefon.

Soluție

Numărul cuvintelor de lungime m peste un alfabet format din n litere este n^m. Cu o literă oarecare x încep n^{m-1} cuvinte.

Dacă prima literă a cuvântului este a k-a din cele n, atunci numărul de ordine al cuvântului este cel puțin $k \cdot n^{m-1}$ și cel mult $(k+1) \cdot n^{m-1} - 1$.

Pentru cuvântul rămas după eliminarea primei litere se aplică aceleași raționamente.

Funcția `Numar(i)` returnează numărul de ordine al unui cuvânt format din ultimele $(m - i + 1)$ litere ale cuvântului inițial, în cadrul mulțimii cuvintelor de lungime $(m - i + 1)$ peste un alfabet care conține primele n litere ale alfabetului.

Listing TELEFON.PAS

```
Program Telefon;
var a:string;
    n,m:Byte;

function N_la_M(i:Byte):Longint;
begin
  if i=0 then N_La_M:=1
         else N_La_M:=n*N_La_M(i-1)
end;

function Numar(i:Byte):Longint;
var j:Byte;
begin
  if i=m+1 then Numar := 1
  else Numar:=(Ord(a[i])-Ord('a'))*
                        N_la_M(m-i)+Numar(i+1)
end;

Begin
  Write('n='); Readln(n);
  Write('m='); Readln(m);
  Write('Numele persoanei:'); Readln(a);
  Writeln('Persoana are numarul de ordine:',
                                    Numar(1))
End.
```

Exemplu

Intrare

```
n=3
m=2
Numele persoanei:bc
```

Ieșire

```
Persoana are numarul de ordine:6
```

Permutări ordonate

Peste mulțimea permutărilor de n elemente se consideră relația de ordine lexicografică. Se dă o permutare cu n elemente și se cere numărul ei de ordine relativ la relația considerată.

Soluție

Pentru o mulțime de n elemente avem în total $n!$ permutări ale elementelor. Considerăm permutările mulțimii {1,2,...,n}.

Numărul permutărilor de lungime n, care încep cu un număr oarecare x din mulțimea {1,2,...,n} este $(n - 1)!$.

Numărul de ordine al unei permutări de lungime n, care începe cu x este cel puțin $(x - 1) \cdot (n - 1)! + 1$ și cel mult $x \cdot (n - 1)!$.

Aceleași raționamente se aplică după eliminarea primei cifre a unei permutări. Trebuie avut grijă, ca permutarea rămasă să fie consistentă. Pentru a realiza acest lucru se scade o unitate din fiecare număr rămas, dacă acesta este mai mare decât cel eliminat.

Listing PERMUTARI.PAS

```
program Permutari;
const MaxN = 10;
var a:array[1..MaxN] of 1..MaxN;
    n,i:Byte;

function factorial(i:Byte):Longint;
begin
  if i=0 then factorial:=1
         else factorial:=i*factorial(i-1)
end;

function Numar(i:Byte):Longint;{ returneaza }
var j:Byte;          { numarul de ordine al }
begin        { permutarii consistente care }
  if i=n+1           { incepe pe pozitia i }
  then Numar:=1
  else
    begin
      for j:=i+1 to n do
        if a[j] > a[i] then Dec(a[j]);
      Numar:=(a[i]-1)*factorial(n-i)+
                                  Numar(i+1)
    end
end;
```

```
Begin
  Write('n='); Readln(n);
  Write('Permutarea:');
  for i:=1 to n do
    Readln(a[i]);
  Writeln('Permutarea are numarul de ',
                        'ordine:',Numar(1));
End.
```

Exemplu

Intrare

```
n=3
Permutarea:2 3 1
```

Ieșire

```
Permutarea are numarul de ordine:4
```

Submulțimi ordonate

Fie submulțimile mulțimii {1,2,...,n}. Acestea se consideră ordonate lexicografic. Se dă o submulțime și se cere numărul ei de ordine.

Soluție

Numărul submulțimilor mulțimii {1,2,...,n} este 2^n (dacă se contorizează și mulțimea vidă și submulțimea maximă, formată din toate elementele mulțimii).

Numărul submulțimilor mulțimii {1,2,...,n} care încep cu un număr oarecare x sunt în număr de 2^{n-x}. Numărul de ordine al unei submulțimi a mulțimii {1,2,...,n} care începe cu x este cel puțin $2^{n-1} + 2^{n-2} + ... + 2^{n-x+1}$ și cel mult $2^{n-1} + 2^{n-2} + ... + 2^{n-x} - 1$. Aceleași raționamente se aplică pentru submulțimea rămasă. Aceasta va trebui să fie o submulțime validă a mulțimii {1,2,...,n-1}. Pentru a realiza acest lucru scădem câte o unitate din fiecare număr al submulțimii rămase. Această operație este posibilă în cazul în care considerăm că elementele unei submulțimi sunt ordonate crescător.

Listing SUBM.PAS

```
program Submultimi;
const MaxN=10;
var a:array[1..MaxN] of 1..MaxN;
    n,m,i:Byte;

function DoiLaN(i:Byte):Longint;
begin          { calculeaza 2 la puterea i }
  DoiLaN:=1 shl i
end;

function Numar(i:Byte):Longint; { returneaza }
var j:Byte;              { numarul de ordine al }
    nr:Longint;  { submultimii consistente }
begin           { care incepe pe pozitia i }
  if i=m+1        { in submultimea initiala }
    then Numar:=1
```

```
    else
      begin
        for j:=i+1 to m do
          Dec(a[j],a[i]);
        nr:=1;
        for j:=1 to a[i]-1 do
          nr:=nr+DoiLaN(n-i-j+1);
        Numar:=nr+Numar(i+1)
      end
end;

Begin
  Write('n='); Readln(n);
  Write('m='); Readln(m);
  Writeln('Submultimea:');
  for i:=1 to m do
    Read(a[i]);
  Writeln('Submultimea are numarul de ',
                          'ordine:',Numar(1))
End.
```

Exemplu

Intrare

```
n=3
m=2
Submultimea:2 3
```

Ieșire

```
Submultimea are numarul de ordine:6
```

Parantezări ordonate

Se consideră mulțimea șirurilor de *n* perechi de paranteze rotunde împerecheate corect. Această mulțime se consideră ordonată invers după poziția pe care se închide paranteza deschisă pe prima poziție.

Regula de ordonare se repetă pentru parantezele rămase.

Se dă un șir de paranteze și se cere numărul său de ordine.

Soluție

Numărul șirurilor de *n* perechi de paranteze împerecheate corect este numărul lui *Catalan*. Acesta are formula

$$T_n = T_0 \cdot T_{n-1} + T_1 \cdot T_{n-2} + ... + T_{n-1} \cdot T_0,$$

unde $T_0 = T_1 = 1$.

Paranteza care se deschide pe prima poziție se închide pe una dintre pozițiile pare (2, 4, 6, ..., 2*n*).

Dacă prima pereche de paranteze se închide pe poziția *k*, atunci numărul de ordine al șirului care conține această configurație este cel puțin $T_{n-1} \cdot T_0 + ... + T_{n-[k/2]-1} \cdot T_{[k/2]}$ și cel mult $T_{n-1} \cdot T_0 + ... + T_{n-[k/2]} \cdot T_{[k/2]+1}$.

În interiorul acestei paranteze se află [*k*/2]-1 paranteze care se închid corect. La dreapta mai rămân [(*n*-*k*)/2] perechi de paranteze care, de asemenea, se închid corect. Pentru aceste două secvențe de paranteze se aplică aceleași raționamente.

Listing PARANT.PAS

```
Program Paranteze;
const MaxN=10;
var a:string;
    catalan:array[0..MaxN] of Longint;
    n:Byte;

procedure Calcul_Catalan;
var k,j:Byte;
begin
  catalan[0]:=1; catalan[1]:=1;
  for k:=2 to n do
  begin
    catalan[k]:=0;
    for j:=0 to k-1 do
      catalan[k]:=catalan[k]+
                  catalan[j]*catalan[k-1-j]
  end
end;

function Numar(i,n:Byte):Longint;
var nr:Longint;          { returneaza numarul }
    dif,k,j:Byte;        { de ordine al sirului }
begin                    { corect de paranteze care }
  if n>0                 { incepe pe pozitia i si are}
  then                   { n perechi de paranteze }
    begin
      nr:=0; dif:=1; k:=1;
      while dif>0 do
      begin
        Inc(k);
        if a[i+k-1]='(' then Inc(dif)
                        else Dec(dif)
      end;
      for j:=n-1 downto k div 2 do
        Inc(nr,catalan[j]*catalan[n-1-j]);
      nr:=nr+Numar(i+1,k div 2 - 1)*
                          catalan[n-k div 2];
      Numar:=nr+Numar(i+k,n-k div 2)
    end
  else Numar:=0
end;

begin
  Write('Parantezarea:'); Readln(a);
  n:=Length(a) div 2;
  Calcul_Catalan;
  Writeln('Parantezarea are numarul de ', '
                      ordine: ',Numar(1,n)+1)
End.
```

Exemplu

Intrare

```
Parantezarea:()(())
```

Ieșire

```
Parantezarea are numarul de ordine:3
```

Reconstituirea unei configurații

Vom exemplifica acum și cazul invers, adică reconstituirea configurației atașate unui număr de ordine dat.

Raționamentele prezentate mai sus se păstrează și în această situație.

Vom trata în continuare cazul permutărilor.

Enunț

Peste mulțimea permutărilor de n elemente se consideră relația de ordine lexicografică. Se dă un număr de ordine și se cere permutarea atașată.

Soluție

Pentru o mulțime de n elemente avem în total $n!$ permutări ale elementelor. Considerăm că permutările sunt din mulțimea {1,2,...,n}. Fie x numărul de ordine al permutării pe care trebuie să o determinăm.

Numărul permutărilor de lungime n, care încep cu un număr oarecare k din {1,2,...,n} este $(n - 1)!$. Trebuie să determinăm numărul k care satisface condiția:

$$(k - 1) \cdot (n - 1)! + 1 \le x < k \cdot (n - 1)!.$$

Pe prima poziție a permutării vom amplasa numărul k astfel determinat. Aceleași raționamente se aplică pentru restul pozițiilor permutării. Pe acestea vor trebui amplasate numere din mulțimea {1,...,k-1,k+1,...,n}.

Listing RECONST.PAS

```
Program ReconstituirePermutare;
const MaxN=10;
var a:array[1..MaxN] of Byte;
    n,i,j,k:Byte;
    x:Longint;
    perm:array[0..MaxN] of Longint;
    folosite:set of Byte;

procedure CalculeazaPerm(n:Byte);
      { calculeaza numarul permutarile de }
      {                1,2,...,n elemente }
begin
  perm[0]:=1;
  for i:=1 to n do
    perm[i]:=perm[i-1]*i
end;

Begin
  Write('n='); Readln(n);
  Write('Numarul de ordine:'); Readln(x);
  CalculeazaPerm(n);
  folosite:=[];
  for i:=1 to n do
  begin
    k:=1;
    while k*perm[n-i]<x do Inc(k);
     { determinam numarul de pe pozitia i }
    Dec(x,(k-1)*perm[n-i]);
    a[i]:=1;
    for j:=1 to n do
    begin
      if not (j in folosite)
        then Dec(k);
      if k=0 then Break
    end;                { permutarea trebuie }
                        { sa fie consistenta }
    a[i]:=j;
    folosite:=folosite+[j];
  end;
  Write('Permutarea este:');
  for i:=1 to n do
    Write(a[i],' ')
End.
```

Exemplu

Intrare

```
n=3
x=4
```

Ieșire

```
Permutarea este: 2 1 3
```

Probleme propuse

Să se reconstituie configurațiile, plecând de la numerele de ordine pentru celelalte probleme prezentate în cadrul acestui articol.

Combinări

Fie mulțimea combinărilor de n elemente luate câte k, peste care se consideră relația de ordine lexicografică. Se dă o combinare și se cere să se determine numărul ei de ordine. Să se reconstituie combinarea care are un număr de ordine dat.

Arbori

Fie mulțimea arborilor parțiali ai unui graf neorientat complet cu n vârfuri, peste care se consideră relația de ordine lexicografică relativă la muchii. Muchia (k, j) este mai mică decât muchia (g, f), dacă $k < g$ sau $k = g$ și $j < f$. Se dă un arbore și se cere să se determine numărul lui de ordine. Să se reconstituie un arbore având un număr de ordine dat.

Parantezări ordonate lexicografic

Se consideră mulțimea șirurilor de n perechi de paranteze rotunde împerecheate corect. Această mulțime este ordonată lexicografic considerând '(' < ')'. Se dă un șir de paranteze și se cere numărul lui de ordine. Să se reconstituie un șir având numărul de ordine dat.

Mihai Oltean și Crina Groșan sunt doctoranzi ai Universității Babeș-Bolyai din Cluj. Pot fi contactați prin e-mail la molteanu@cs.ubbcluj.ro, *respectiv* cgrosan@cs.ubbcluj.ro.

Algoritmi de TARE-PARCURGERE

Radu Cristian Vișinescu

Acest articol introduce conceptul de tare-parcurgere a unui graf. În acest sens, sunt descrise adaptări ale unor algoritmi de parcurgere cunoscuți cum ar fi BF, D sau DF; în final se introduce parcurgerea DL (Deapth Last). Nu în ultimul rând, acest articol este o "pledoarie în favoarea lui Tezeu".

Problema labirintului

Considerăm un labirint ca fiind o construcție pătratică formată din camere (celule). Un labirint se poate codifica printr-o matrice A de ordin n în care intrarea a_{ij} este un număr între 0 și 15 cu următoarea semnificație: *dacă se consideră vecinii camerei (i, j) in ordinea nord, est, sud, vest și se reține 1 în cazul existenței unei uși spre vecinul respectiv și 0 în caz contrar; numărul binar de patru cifre astfel format este convertit în baza 10.*

Observație

O ușă se poate deschide în ambele direcții, deci nu orice tablou pătratic având ca elemente numere între 0 și 15 codifică un labirint; elementele tabloului nu sunt total independente.

Unii consideră că problema ieșirii dintr-un labirint nu se poate rezolva în timp polinomial. De exemplu, într-un articol în care era vorba de dificultatea găsirii unor algoritmi eficienți în inteligența artificială, pentru rezolvarea automată de probleme era folosită în mod eronat următoarea analogie: *dacă operația de verificare a corectitudinii unei demonstrații este analoagă verificării dacă un drum dat asigură ieșirea din labirint, găsirea unei demonstrații corespunde construirii unui astfel de drum.*

Abordarea prin teoria grafurilor

Vom arăta că putem găsi un drum de ieșire din labirint într-un timp $O(n^2)$. Mai mult, determinarea drumului de lungime minimă de ieșire dintr-un labirint poate fi realizată și ea într-un timp $O(n^2)$.

Distanța dintre camere se presupune a fi constantă și egală cu 1.

Un labirint este de fapt un graf cu n^2 vârfuri și cu mai puțin de $4n^2$ muchii. Drept noduri considerăm camerele iar drept muchii ușile dintre camere. O parcurgere DF a acestui graf se efectuează într-un timp $O(n^2)$, aplicarea acestui algoritm asigură deja găsirea unei soluții într-un timp $O(n^2)$. Algoritmul de parcurgere DF este următorul:

```
algoritm DF(i)
  vizitează nodul i
  marcat(i)← adevărat
  pentru toți vecinii j ai lui lui i pentru care
                          marcat(j)=fals execută
    DF(j)
  sfârșit pentru
```

Inițializăm marcajele cu valoarea 0 și apelăm procedura pentru camera de plecare.

Pe același graf putem aplica o parcurgere tip BF în care vom determina în paralel distanțele minime de la camera de plecare și toate celelalte camere (algoritmul lui *Lee*). Oprim execuția parcurgerii BF în momentul în care ajungem într-o cameră de ieșire.

Algoritmul lui Lee este prezentat în continuare:

```
algoritm Lee(i)
  Coada ← vidă
  STIVA ← vidă
  vizitează nodul i
  marcaj(j) ← adevărat
  d ← 0
  Coada ← (i,0,d)
  cât timp Coada nu este vidă execută
    (i,pred_i,d) ← Coada
```

```
    STIVA ← (i,pred_i)
    pentru toți vecinii j ai lui lui i pentru care
                          marcat(j)=fals execută
      vizitează nodul j
      marcaj(j) ← adevărat
      Coada ← (j,i,d+1)
    sfârșit pentru
  sfârșit cât timp
```

În acest caz inițializăm, de asemenea, marcajele cu 0 și apelăm procedura pentru camera de plecare.

Ordinul de complexitate al acestei parcurgeri BF este liniar în numărul de noduri ale grafului, deci este $O(n^2)$ în cazul problemei de față.

Drumul de la prima cameră de ieșire găsită la camera de plecare se poate determina printr-o singură parcurgere în sens invers a vectorului STIVA. Acest vector are dimensiunea cel mult n^2.

Să analizăm mai atent algoritmul:

- Coada va conține o listă de noduri vizitate.
- Fiecare nod este vizitat și intră în coadă o singură dată.
- Prelucrarea unei camere, cu alte cuvinte vizitarea vecinilor nemarcați, se execută la extragerea din coadă. În final în STIVA se vor afla toate nodurile prelucrate până în momentul găsirii camerei de ieșire, fiecare dintre ele având legătura înapoi către camera de pornire (prin concatenerea acestor legături se determină drumul cel mai scurt).

În concluzie, problema găsirii drumului optim de ieșire este rezolvată folosind algoritmul lui *Lee*, tot într-un timp $O(n^2)$.

În continuare vom rafina analiza complexității notând cu n' numărul de noduri care pot fi efectiv parcurse (cele din componenta conexă care conține camera de pornire) și cu m' numărul de muchii ale acestei componente conexe. Ambii algoritmi prezentați anterior au ordinul de complexitate $O(n'+m')$.

Observație

Algoritmul "clasic" de găsire a tuturor ieșirilor din labirint (dat spre exemplu în Tudor Sorin, *Tehnici de programare*) poate determina prima soluție după un timp exponențial.

Problema labirintului pe cazul real

Problema reală a omului prins în labirint este de fapt o reformulare a problemei inițiale și anume transformarea ei într-o problemă cu informație incompletă.

Labirintul este "dat" în întregime la început, dar omul prins în labirint nu dispune de întreaga sa hartă. În camera de pornire nu cunoaște la început decât existența a cel mult patru uși de legătură cu vecinii direcți (nord, est, sud și vest). Nu va cunoaște existența unei uși între nodurile i și j până când nu ajunge fie în camera i, fie în camera j.

De aici rezultă că la aplicarea oricărui algoritm de ieșire din labirint trebuie impusă condiția ca această parcurgere să fie efectuată "din cameră în cameră".

Cu alte cuvinte, în analiza complexității acestor algoritmi este natural să considerăm ca operație fundamentală trecerea dintr-o cameră într-o cameră vecină.

În cazul parcurgerii DF numărul de operații fundamentale efectuate coincide cu numărul de vizitări. Dacă renunțăm la optimalitate, prin acest algoritm găsim o ieșire din labirint într-un timp $O(n')$.

Vom arăta că aplicarea algoritmului lui *Lee* duce la un timp de ordinul $O(n'^2)$.

Prin acest algoritm putem reține în permanență, pentru fiecare cameră vizitată, lanțul care o unește cu camera de pornire.

Acest lucru poate fi realizat eliminând stiva, și introducând în coadă fiecare nod vizitat împreună cu întreg lanțul care îl unește cu nodul de plecare.

Deoarece următoarea cameră care trebuie prelucrată este prima de la ieșirea din coadă, pentru a o putea accesa este suficient să parcurgem lanțul din camera curentă (ultima prelucrată) în camera de plecare și apoi din camera de plecare în camera care trebuie prelucrată.

Distanța maximă dintre oricare două camere este $n' - 1$, deci acest pas de parcurgere "din cameră în cameră" este realizat în maximum $2n'$ pași. Pasul trebuie repetat pentru fiecare dintre cele n' camere. În concluzie, complexitatea în timp devine $O(n'^2)$, în raport cu operația fundamentală considerată.

Vom descrie acum un labirint pentru care aplicarea parcurgerii BF astfel modificată duce la un timp pătratic față de mulțimea efectivă a nodurilor. Considerăm de la sud la nord lanțul de camere 1, 2, ..., n',camera de pornire fiind n', iar spre est lanțul $n' + 1$, $n' + 2$, ..., $2n'$. Conform algoritmului vom trece din camera $n' - i$ în camera $n' + i$ pentru fiecare i. Suma deplasărilor va fi $2(1 + 2 + \ldots n')$, deci ordinul de complexitate va fi $O(n'^2)$.

Consumul de memorie suplimentară este foarte mare; omul prins în labirint trebuie să rețină în permanență date referitoare la subgraful deja parcurs.

Iată două argumente care conduc și ele către alegerea parcurgerii DF.

Concret, în plan, aceasta revine la următoarele: la fiecare intersecție se alege drumul cel mai din stânga și, în plus, trebuie să existe un "fir al *Ariadnei*" care leagă în permanență camera curentă de camera de plecare. Vom prezenta și o condiție suficientă pentru a ne putea dispensa de acest "fir al *Ariadnei*", precum și motivul pentru care acesta este necesar în cazul general.

Cazul labirintului arbore

În continuare vom considera că grafurile care reprezintă labirintul sunt "scufundate" în plan. Astfel vom putea da o orientare muchiilor din fiecare cameră de intersecție, deoarece planul poate fi considerat ca fiind orientat. Corectitudinea următorilor algoritmi se bazează pe astfel de raționamente de orientare.

Dacă labirintul reprezintă un graf care este și arbore, atunci în parcurgerea sa putem face abstracție de conținu-

tul stivei. De fapt, vom descrie un algoritm de parcurgere în plan a unui arbore fără memorie suplimentară. Ideea de bază este de a alege la fiecare intersecție cel mai din stânga drum (care, la limită, este ușa pe care tocmai am intrat).

```
algoritm arbore(i)
  i0 ← i
  scrie i
  d ← N
  repetă
    d ← succ(succ(succ(succ(d))))
                        /* "prima la stânga" */
    cât timp nu există nod pe direcția d execută
      d ← succ(d)
    sfârșit cât timp
    j ← un nod de pe direcția d
    i ← j
  scrie i
  până când i=i0
```

Observații

- Considerăm cele patru direcții nord, este, sud și vest și definim funcția succ astfel încât succ(nord)=est; succ(est)=sud; succ(sud)=vest; succ(vest)=nord. Arborele va avea cel puțin două noduri.
- În cazul existenței unui posibil ciclu în labirint acest ultim algoritm poate eventual intra într-un ciclu infinit. Așadar acesta este un algoritm euristic.
- Există camere vizitate de mai multe ori. O vizitare este indicată de un apel **scrie**.
- În cazul real ordinul de complexitate este $O(n')$

Algoritmul folosit de Tezeu

Așa cum am mai spus, din punctul de vedere al omului prins în labirint, problema este cu informație incompletă. Deoarece labirintul în care a fost prins *Tezeu*, nu era neapărat de tip arbore, el a avut nevoie de "firul *Ariadnei*" și a ales cu siguranță parcurgerea DF.

Ca memorie suplimentară este folosit doar un vector de marcaj de lungime n' (elementele marcate constituie "firul *Ariadnei*"). Algoritmul este următorul:

```
algoritm Tezeu(i)
  vizitează nodul i
  marcaj_vizit(i) ← adevărat
  scrie i
  pred ← 0
  marcaj(i) ← adevărat
  repetă
    d ← succ(succ(succ(succ(d))))
                        /* prima la stânga */
    cât timp nu există nod pe direcția d sau există nod
        marcat pe direcția d și nodul marcat de pe
            direcția d nu este predecesorul nodului
                                    curent execută
      d ← succ(d)
    sfârșit cât timp
    j ← un nod de pe direcția d
    dacă nu marcaj_vizit(i) atunci
      vizitează nodul i
      marcaj_vizit(i) ← adevărat
    sfârșit dacă
    dacă j=pred atunci marcaj(i) ← fals
                altfel marcaj(i) ← adevărat
    sfârșit dacă
    i ← j
    scrie i
    dacă există p vecin al lui i și marcaj(p)
      atunci pred ← p
      altfel stop
    sfârșit dacă
  până când fals
```

Să analizăm mai atent algoritmul:

- Pentru fiecare nod după vizitare determinăm predecesorul acestuia ca fiind acel unic nod vecin care este și marcat.
- Singurul nod care nu are predecesor în acest punct va fi nodul de pornire (este totuși și el marcat).
- După calculul predecesorului prin tehnica "prima la stânga" calculăm succesorul nodului i. Trebuie tratate două cazuri: *succesorul coincide sau nu cu predecesorul.* Aceste cazuri sunt tratate corespunzător actualizării "firului *Ariadnei*".

Observații

- Ca memorie suplimentară se folosesc doar doi vectori de biți.
- În raport cu operația fundamentală algoritmul are ordinul de complexitate $O(n')$, producând la ieșire un circuit cu $2(n - 1)$ muchii, circuit ce parcurge arborele DF.
- Circuitul tare-parcurgerii este listat prin apeluri **scrie**.

Considerații de teoria grafurilor

Problemele de parcurgere prezentate se extind natural la cazul unui graf conex. Acest graf este dat prin listele de adiacență (liste circulare), pentru fiecare nod reținându-se și gradul său.

Așadar, găsirea succesorului după regula "prima la stânga" poate fi ușor generalizată.

Problema reală a labirintului conduce în mod natural la următoarea noțiune de tare-parcurgere a unui graf:

Definiție

Fie un graf neorientat $G = (V, M)$. Vom numi o tare-parcurgere a sa un circuit care trece cel puțin o dată prin toate vârfurile grafului G, împreună cu o alegere pe acest circuit a mulțimii vârfurilor lui G.

Observație

Noțiunea de parcurgere de până acum însemna o liniarizare (listare sau altă operație) a nodurilor grafului, așadar o tare-parcurgere determină o parcurgere prin nodul său de

plecare. Vom spune că tare-parcurgerea respectivă este conformă cu parcurgerea determinată de ea.

Să rezumăm:

- În primii doi algoritmi vizitarea nodurilor poate fi pusă în corespondență cu listarea circuitului corespunzător unei tare-parcurgeri. Prin utilizarea unui vector suplimentar de marcaj care are la început toate valorile setate pe 0, și executarea unei operații asupra lui la prima vizitare (de exemplu, marcarea sa ca nod ales pe circuit), se completează generarea unei tare-parcurgeri conform definiției date. În concluzie, parcurgerea DF poate fi imediat adaptată să furnizeze o tare-parcurgere conformă cu parcurgerea propriu-zisă. Algoritmul nu este optim în raport cu operația fundamentală (furnizează o tare-parcurgere de circuit de lungime minimă).
- În cele prezentate anterior am adaptat algoritmul BF (*Lee*) care acum poate furniza o tare-parcurgere conformă cu parcugerea clasică, în ordinea distanțelor minime de la nodul de plecare. Complexitatea este $O(n'^2)$ în raport cu operația fundamentală.
- Ne punem problema dacă orice algoritm de parcurgere poate fi adaptat pentru a genera o tare-parcurgere conformă cu respectiva parcurgere.

Cazul algoritmului de parcurgere D (Depth)

Vă prezentăm în continuare descrierea în pseudocod a algoritmului clasic. Această descriere îi aparține d-lui *prof. dr. Horia Georgescu.*

```
algoritm D(i)
  vizitează nodul i
  marcat(i) ← adevărat
  STIVA ← vidă
  repetă
    pentru toți vecinii j ai lui lui i pentru care
                          marcat(j)=fals execută
      vizitează nodul j
      marcat(j) ← adevărat
      STIVA ← j
    sfârșit pentru
    dacă STIVA=vidă atunci stop
                      altfel STIVA←i
      sfârșit dacă
  până când fals
```

Observații

- Stiva conține o listă de noduri vizitate și neprelucrate
- La fiecare pas al buclei **repetă** prelucrăm un nod curent i; în acest moment toate nodurile aflate în stivă au proprietatea că sunt legate de un nod din drumul de la nodul de plecare la nodul i, drum care va fi memorat, pentru simplitate, în altă stivă auxiliară.
- Algoritmul are ordinul de complexitate $O(n'+m')$

Vă prezentăm în continuare descrierea în pseudocod a adaptării sale pentru a genera o tare-parcurgere conformă.

```
algoritm D_tare(i)
  vizitează nodul i
  marcat(i) ← adevărat
  STIVA ← vidă
  S_AUX ← i
  scrie i
  repetă
    pentru toți vecinii j ai lui lui i pentru care
                          marcat(j)=fals execută
      vizitează nodul j
      marcat(j) ← adevărat
      STIVA ← j
    sfârșit pentru
    S_AUX ← vârful stivei STIVA
    scrie vârful stivei STIVA
    dacă STIVA=vidă atunci stop
    altfel
      STIVA ← i
      repetă
        S_AUX ← x
        scrie x
      până când x este vecin cu i;
      S_AUX ← x
    sfârșit dacă
  până când fals
```

Concluzii

Adaptarea algoritmului D generează la rândul său o tare-parcurgere de circuit de lungime minimă. De fapt, tare-parcurgerea generată de ambii algoritmi este identică din punct de vedere al circuitului parcurs.

De ce a ales *Tezeu* algoritmul DF în locul algoritmului D? Din punctul său de vedere ambii algoritmi sunt echivalenți ca ordin de complexitate. Probabil că în labirint era întuneric și nu putea să marcheze sau să memoreze camerele. Aceasta revine la minimizarea memoriei suplimentare.

Dăm în încheiere o definiție a unui labirint ca fiind un *graf cu informație incompletă.*

Definiție

Numim labirint un graf neorientat G = (V, M) împreună cu o submulțime V_p de vârfuri distincte numite noduri de plecare.

Bibliografie

1. Atanasiu A., *Culegere de probleme Pascal,* Ed. Petrion
2. Knuth D.E., *Algoritmi fundamentali,* Ed. Tehnică - 1976
3. D.E.Knuth, *Sortare și căutare,* Ed. Tehnică - 1976
4. Gazeta Matematică, colecție
5. PC REPORT, colecție
6. Leon Livovschi, Horia Georgescu, *Bazele informaticii - Algoritmi. Elaborare și complexitate*

Domnul profesor Radu Cristian Vișinescu este cadru didactic la Liceul "Ion Luca Caragiale" din Ploiești. Poate fi contactat prin e-mail la adresa raduv@lphilc.sfos.ro.

Măsurarea TIMPULUI

Mihai Scorțaru

Mulți programatori doresc să măsoare sau să controleze durata unei operațiuni realizate de calculator. Chiar și la concursurile de programare, timpul alocat rulării este limitat, motiv pentru care ar fi bine ca programul să "știe" când timpul de execuție se apropie de sfârșit. În acest articol vă vom prezenta câteva modalități de a utiliza ceasul sistem al calculatorului.

Preliminarii

Comportamentul multor programe depinde de un anumit moment de timp. Astfel, un "ceas deșteptător" virtual va putea suna la ora stabilită, un satelit artificial va putea fi lansat atunci când trebuie etc.

Din aceste motive, calculatoarele trebuie să știe în orice moment "cât e ceasul". Unele programe trebuie să știe cât timp a trecut de la începerea rulării. Ne putem imagina oricâte exemple din robotica industrială sau din lumea burselor, dar pentru cititorii *GInfo* cel mai potrivit exemplu este cel al programelor care rezolvă diferite probleme date la concursurile de programare. De foarte multe ori, atunci când concurenții nu găsesc un algoritm eficient pentru rezolvarea unei probleme, ei preferă să utilizeze diverse metode pentru a testa diferite soluții posibile și a o alege pe cea mai bună. Programele vor încerca să găsească soluții noi până când timpul de execuție admis se va apropia de sfârșit și apoi vor furniza cea mai bună soluție găsită până în acel moment.

Pentru a realiza acest lucru, este folosită (prin diferite procedee) întreruperea `08h` a sistemului de operare *MS DOS*, numită și întreruperea *timer*-ului. Vom exemplifica aceste procedee pentru compilatoarele *Borland Pascal 7.0* și *Borland C++ 3.1*.

Operația folosită pentru exemplificarea măsurării timpului scurs este executarea unui ciclu vid având două miliarde de iterații.

Subprograme de tip GetTime

Cea mai simplă astfel de metodă este folosirea funcțiilor care returnează, într-o anumită formă, timpul din momentul în care sunt apelate. În *Pascal* avem procedura `GetTime` a *unit*-ului `DOS`, iar în C++ funcția `gettime()` declarată în fișierul *header* `DOS.H`.

Pentru *Pascal* va trebui să folosim un apel de tipul `GetTime(Ore,Minute,Secunde,Sutimi)`, unde `Ore`, `Minute`, `Secunde` și `Sutimi` sunt variabile de tip `Word` sau un tip compatibil cu acesta. Denumirile pe care le-am ales pentru cele patru variabile sugerează destul de clar semnificațiile valorilor pe care le vor primi după apel.

Pentru C++ apelul folosit va fi de tipul `gettime(&t)`, unde `t` este o structură de tip `time`.

Structura `time` este definită în fișierul *header* `DOS.H` astfel:

```
struct time{
  unsigned char ti_min; // minute
  unsigned char ti_hour; // ore
  unsigned char ti_hund; // sutimi de secunda
  unsigned char ti_sec; // secunde
};
```

Semnificațiile valorilor pe care le vor primi cele patru câmpuri după apelul funcției apar în comentarii.

Pentru a măsura exact timpul care a trecut de la începerea execuției programului, prima instrucțiune executată va trebui să fie un apel al unei proceduri/funcții de tip `gettime`, urmată de salvarea în variabile auxiliare a valorilor obținute.

În momentul în care dorim să știm cât timp a trecut de la începerea execuției programului, vom realiza un nou apel și vom calcula timpul care a trecut între primul și ultimul apel, efectuând câteva operații simple.

Varianta Pascal

Listing GETTIME.PAS

```
uses dos;
var ore_start,minute_start,secunde_start,
    sutimi_start,ore_final,minute_final,
    secunde_final,sutimi_final,ore,
    minute,secunde,sutimi:Word;
    i:Longint;
```

```
function Zerouri(nr:Word):string;
var s:string;
begin
  Str(nr,s);
  if Length(s)=1 then s:='0'+s;
  Zerouri:=s
end;

Begin
  Gettime(ore_start,minute_start,
                secunde_start,sutimi_start);
  for i:=1 to 200000000 do;
  Gettime(ore_final,minute_final,
                secunde_final,sutimi_final);
  sutimi:=100+sutimi_final-sutimi_start;
  Dec(secunde_final,1-sutimi div 100);
  sutimi:=sutimi mod 100;
  secunde:=60+secunde_final-secunde_start;
  Dec(minute_final,1-secunde div 60);
  secunde:=secunde mod 60;
  minute:=60+minute_final-minute_start;
  Dec(ore_final,1-minute div 60);
  minute:=minute mod 60;
  ore:=(24+ore_final-ore_start) mod 24;
  Writeln('Timp de executie: ',Zerouri(ore),
       ' ore ', Zerouri(minute),' minute ',
                          Zerouri(secunde),'.',
                Zerouri(sutimi),' secunde.')
End.
```

Varianta C++

Listing GETTIME.CPP

```
#include <stdio.h>
#include <dos.h>
void main(void){
  struct time t,tstart;
  int ore,minute,secunde,sutimi;
  gettime(&tstart);
  for (long i=0;i<2000000000;i++);
  gettime(&t);
  sutimi=100+t.ti_hund-tstart.ti_hund;
  t.ti_sec-=1-sutimi/100;
  sutimi%=100;
  secunde=60+t.ti_sec-tstart.ti_sec;
  t.ti_min-=1-secunde/60;
  secunde%=60;
  minute=60+t.ti_min-tstart.ti_min;
  t.ti_hour-=1-minute/60;
  minute%=60;
  ore=(24+t.ti_hour-tstart.ti_hour)%24;
  printf("Timp de executie: %02d ore %02d \
          minute %02d.%02d secunde.\n",
                ore,minute,secunde,sutimi);
 }
```

Această metodă este cea mai ineficientă, datorită timpului relativ mare necesar executării subprogramelor de acest tip. În plus, se pierde timp atât la execuție, cât și la scrierea codului pentru calcularea efectivă a diferenței în ore, minute, secunde și sutimi.

Subprograme de tip SetTime

Pierderea de timp datorată calculării efective a diferenței în ore, minute, secunde și sutimi poate fi evitată setând la început ceasul sistem la ora `00:00`, apoi pentru citirea timpului scurs între momentul setării ceasului pe 0 și momentul dorit, vom apela subprogramele prezentate anterior.

Într-un program *Pascal* va trebui să folosim procedura `SetTime` a *unit*-ului `DOS` prin apelul `SetTime(0,0,0,0)`.

Într-un program C++ vom folosi funcția `settime()`, declarată în fișierul *header* **`DOS.H`**. Apelul trebuie să fie de tipul `settime(&t)`, unde `t` este o structură de tip `time`. Câmpurile variabilei `t` trebuie să aibă toate valoarea 0.

Dacă am setat la început ceasul sistem la ora `00:00`, atunci un apel `GetTime` ne va da informații referitoare la timpul scurs de la începerea execuției. Nu mai este nevoie de efectuarea de operații suplimentare pentru determinarea timpului în ore, minute, secunde și sutimi.

Varianta Pascal

Listing SETTIME.PAS

```
uses dos;
var ore,minute,secunde,sutimi:Word;
    i:Longint;
function Zerouri(nr:Word):string;
var s:string;
begin
  Str(nr,s);
  if Length(s)=1 then s:='0'+s;
  Zerouri:=s
end;

Begin
  Settime(0,0,0,0);
  for i:=1 to 2000000000 do;
  Gettime(ore,minute,secunde,sutimi);
  Writeln('Timp de executie: ',Zerouri(ore),
       ' ore ', Zerouri(minute),' minute ',
                          Zerouri(secunde),'.',
                Zerouri(sutimi),' secunde.')
End.
```

Varianta C++

Listing SETTIME.CPP

```
#include <stdio.h>
#include <dos.h>
void main(void){
  struct time t={0,0,0,0};
```

```
  settime(&t);
  for (long i=0;i<2000000000;i++);
  gettime(&t);
  printf("\nTimp de executie: %02d ore %02d \
      minute %02d.%02d secunde\n",t.ti_hour,
              t.ti_min,t.ti_sec,t.ti_hund);
}
```

Această metodă este puțin mai eficientă decât cea anterioară, dar necesită modificarea ceasului sistem, operație care nu este recomandabilă.

Capturarea întreruperii 08h

Această metodă este mai rapidă și nu are efecte secundare nedorite. *Timer*-ul este o rutină care este apelată cu o frecvență de aproximativ 18,2 ori pe secundă, adică o dată la aproximativ 55 de milisecunde. Operația efectuată de această rutină este incrementarea ceasului sistem cu 55 ms. Totuși, prin capturarea întreruperii, putem adăuga propriul nostru cod care să se execute în momentul în care este apelată rutina corespunzătoare întreruperii.

Trebuie doar să fim atenți ca rutina scrisă de noi să apeleze și vechea rutină deoarece, în caz contrar, timpul sistemului se va opri și acest lucru ar putea avea efecte nedorite.

În continuare vom prezenta un exemplu care folosește această metodă pentru a măsura timpul. Să presupunem că dorim să executăm un ciclu timp de 10 secunde; pentru aceasta vom inițializa o variabilă `time` cu valoarea 182 (`10·18.2=182`) și vom decrementa această valoare cu o unitate la fiecare execuție a întreruperii `08h`. Ciclul se va întrerupe în momentul în care variabila `time` va avea valoarea 0.

Într-un program *Pascal* vom folosi procedura `GetIntVec` pentru a captura o întrerupere; această procedură este declarată astfel:

`GetIntVec(IntNo:Byte;var Vector:Pointer)`,

unde `IntNo` indică numărul întreruperii, iar `Vector` va indica adresa la care era păstrată rutina care este executată în momentul apelării întreruperii. Pentru a executa o anumită rutină în momentul apelării întreruperii vom folosi procedura `SetIntVec` a cărei declarație este

`SetIntVec(IntNo:Byte;Vector:Pointer)`,

unde `IntNo` indică numărul întreruperii, iar `Vector` indică adresa la care este păstrată rutina care va fi executată în momentul apelării întreruperii. Aceste două proceduri sunt definite în unit-ul `DOS`.

Într-un program C++ vom folosi funcția `getvect()` pentru a captura o întrerupere; antetul acestei funcții este

`void interrupt (*getvect(int interruptno))()`,

unde `interruptno` indică numărul întreruperii. Funcția va returna un pointer către rutina care este executată în momentul apelării întreruperii. Pentru a executa o altă rutină în momentul apelării întreruperii vom folosi funcția `setvect()` al cărei antet este

```
void setvect(int interruptno,void interrupt
                    (*isr)()),
```

unde `interruptno` indică numărul întreruperii, iar `isr` indică adresa la care este păstrată funcția care va fi executată în momentul apelării întreruperii. Aceste două funcții sunt declarate în fișierul *header* **DOS.H**.

Varianta Pascal

Listing INTR8.PAS

```
uses Dos;
var i,time:Longint;
    OldTimer:procedure;

procedure MyTimer;interrupt;
begin
  Dec(time);
  inline ($9C);
  OldTimer
end;

Begin
  time:=182;
  i:=0;
  GetIntVec(8,@OldTimer);
  SetIntVec(8,@MyTimer);
  while time>0 do Inc(i);
  SetIntVec(8,@OldTimer)
End.
```

Varianta C++

Listing INTR8.CPP

```
#include <dos.h>

long i,time=182;
void interrupt (*OldTimer)(...);

void interrupt MyTimer(...){
  time--;
  OldTimer();
}

void main(void){
  OldTimer=getvect(8);
  setvect(8,MyTimer);
  while(time)i++;
  setvect(8,OldTimer);
}
```

Și această metodă are unele dezavantaje deoarece implică scrierea de cod suplimentar.

În plus, sunt efectuate operații suplimentare cum ar fi decrementarea variabilei `time` sau apelurile rutinei vechi de tratare a *timer*-ului.

Acestea reduc puțin timpul care este folosit pentru efecturea operațiilor pe care trebuie să le realizeze programul.

Locația de memorie $0000:$046C

Ultima variantă pe care o vom prezenta elimină și aceste ultime deficiențe. Ea se bazează pe accesarea directă a unei locații de memorie în care este păstrat numărul de apeluri ale *timer*-ului începând cu ora 00:00.

Numărul de tacți (apeluri ale *timer*-ului) care au trecut de la miezul nopții este reprezentat pe patru bytes și se află la adresa $0000:$046C.

Pentru a utiliza această metodă vom păstra valoarea stocată la această locație în momentul începerii execuției și apoi, pentru a măsura intervalul de timp scurs, vom scădea valoarea inițială din valoarea curentă de la această locație. Diferența va indica numărul de tacți care au trecut de la începerea execuției programului.

În *Pascal*, pentru a accesa direct o locație de memorie putem folosi clauza **absolute**.

În C++, pentru a realiza același lucru, putem folosi *macro*-ul MK_FP definit în cadrul fișierului *header* DOS.H. Acest *macro* creează un pointer din părțile sale componente (segment și deplasament). În cazul nostru segmentul va avea valoarea 0, iar deplasamanetul va avea valoarea 0x46C.

Varianta Pascal

Listing ABSOLUTE.PAS

```
var i,t:Longint;
    time:Longint absolute $0:$46C;

Begin
  t:=time;
  i:=0;
  while (time-t<182) do Inc(i)
End.
```

Varianta C++

Listing MKFP.CPP

```
#include <dos.h>
void main(void){
  long i=0,t,*time=(long*)MK_FP(0,0x46C);
  t=*time;
  while (*time-t<182) i++;
}
```

Chiar și această metodă are un mic dezavantaj; ea nu va da rezultatele așteptate în cazul în care ceasul sistem va deveni 00:00 în timpul execuției programului. Apariția acestei situații este destul de improbabilă în cazul în care se lucrează cu intervale de timp de ordinul secundelor sau minutelor, dar ea ar putea apărea dacă intervalele de timp măsurate sunt mai mari. În acest caz s-ar putea ca programele să nu mai funcționeze conform așteptărilor.

Mihai Scorțaru este redactor-șef adjunct al GInfo. Poate fi contactat prin e-mail la adresa skortzy@xnet.ro.

M/F?

Multe persoane s-au întrebat dacă numeroasele instrumente pe care le folosim zi de zi trebuie considerate a fi de genul feminin sau de genul masculin.

Unele obiecte au primit deja un gen. De exemplu, serialul Star Trek a impus genul feminin pentru navele stelare. Se pune acum problema genului calculatoarelor.

Un grup de oameni de știință (format doar din bărbați) au ajuns la concluzia că trebuie să considerăm calculatoarele ca fiind de genul feminin și au dat cinci argumente în favoarea acestui fapt.

- Nimeni, cu excepția Creatorului lor, nu le înțelege logica internă.
- Limbajul nativ pe care îl folosesc pentru comunicarea cu alte calculatoare nu mai poate fi înțeles de nimeni altcineva.
- Mesajul "Bad command or file name" este la fel de informativ ca și mesajul "Dacă nu știi de ce sunt supărată pe tine, atunci cu siguranță îți voi spune eu!".
- Chiar și cele mai mici greșeli pe care le facem sunt stocate în memoria permanentă pentru a putea fi regăsite ulterior.
- Imediat după ce te atașezi de unul, îți dai seama că cheltui jumătate din venituri cumpărând accesorii pentru el.

Un alt grup (format numai din femei) au ajuns la concluzia că genul calculatoarelor trebuie să fie masculin și au adus și ele cinci argumente în favoarea acestei opinii.

- Au o mulțime de date, dar nu au nici o idee despre ceea ce se întâmplă în jur.
- Ar trebui să te ajute să îți rezolvi problemele dar, în cea mai mare parte a timpului, constituie adevărata problemă.
- Imediat după ce te atașezi de unul observi că, dacă ai mai fi așteptat puțin, ai fi putut obține un model mai bun.
- Pentru a le atrage atenția trebuie neapărat aprinse.
- Descărcări mari de energie la fac inutilizabile pentru restul nopții.

Rezolvarea unei ECUAȚII cu COEFICIENȚI numere întregi mari

Laviniu Bejenaru

Problema rezolvării ecuațiilor poate ridica dificultăți majore, mai ales dacă ea a apărut ca subproblemă în rezolvarea unei probleme practice, care, de obicei, este complexă. Imaginați-vă un inginer care construiește un pod, dar calculează greșit anumiți parametri proveniți tocmai din ecuații, sau un centru spațial unde se realizează trimiterea pe orbită a unor sateliți sau a unor navete. În sensibilul domeniu economic un calcul greșit al rădăcinilor unor ecuații matematice ce modelează diverși parametri economici poate avea urmări sociale nedorite.

Prezentare generală

MAPLE V este un program care permite rezolvarea a numeroase probleme matematice, de la calculul unor rădăcini simple și produse de matrice până la reprezentarea tridimensională a unor funcții complexe în mai multe variabile. Acest produs se bucură de succes în mediul universitar și, în general, în rândul specialiștilor de orice natură, de la ingineri la profesori.

Pachete de funcții

În rezolvarea ecuațiilor o importanță foarte mare o are posibilitatea determinării rădăcinilor acestora cu o anumită precizie și pentru coeficienți numere întregi sau reale, foarte mari, de ordinul miliardelor sau sutelor de miliarde, ceea ce fără ajutorul calculatorului ar fi aproape imposibil de realizat.

Pentru a efectua aproape orice calcul întâlnit în matematică, *Maple V* dispune de numeroase funcții predefinite. Aceste funcții sunt incluse în pachete speciale. Pentru a putea apela o funcție inclusă în pachetul *nume_pachet*, vom folosi comanda:

```
with(nume_pachet);
```

sau

```
with(nume_pachet):
```

Prezența caracterului `';'` la sfârșitul comenzii are ca efect afișarea tuturor funcțiilor ce se află în pachetul respectiv. Prezența caracterului `':'` la sfârșitul comenzii are ca efect doar încărcarea elementelor specifice pachetului respectiv, inhibând afișarea elementelor componente. Un exemplu de pachet este `linalg` care poate fi folosit pentru a rezolva subprobleme clasice de algebră liniară. Pachetul conține numeroase funcții, care permit realizarea unei multitudini de operații de la produsul de matrice (funcția `innerprod`) și calculul determinanților (funcția `det`), până la calculul laplacianului (funcția `laplacian`) ori al unui bloc al formei *Jordan* a unei matrice (funcția `JordanBlock`).

Un alt exemplu de pachet este `geometry`, încărcat prin comanda `with(geometry);` sau `with(geometry):` și care permite efectuarea unor operații cu elemente specifice geometriei. Se pot efectua numeroase operații de la determinarea ariei triunghiurilor (funcția `area`) sau a bisectoarelor unghiurilor (funcția `bisector`) până la determinarea medianelor unui triunghi (funcția `median`). De asemenea, se pot determina ortocentre (funcția `orthocenter`), raze ale unui cerc (funcția `radius`), proiecții (funcția `projection`), sau tangente la un cerc (funcția `tangent`).

Elementele de matematici superioare, cum ar fi cazul ecuațiilor diferențiale, au pachete speciale, de exemplu

DEtools. Se întâlnesc, de asemenea, elemente de teoria grupurilor, teoria numerelor sau statistică.

Aplicație

Vom prezenta acum modul în care pot fi *determinate rădăcinile unor ecuații care au coeficienții foarte mari*, imposibil de calculat în maniera ecuației de gradul II, adică pe hârtie, cu un creion și socotind, folosind o formulă simplă.

Vom folosi pachetul GaussInt, încărcându-l în prealabil prin comanda **with**(Gaussint); Efectuăm comanda

```
nume_exp:=expr;
```

unde nume_exp este un identificator în care se păstrează ecuația, iar expr este expresia ecuației într-o anumită nedeterminată, de exemplu x. Dacă dorim ca în expresia ecuației să folosim identificatori pentru a desemna coeficienții ei, în prealabil trebuie să îi definim în mod similar unor constante simbolice. Produsul dintre coeficienți și termenii care conțin puteri ale nederminatei este notat prin caracterul '*'. Pentru "a *la puterea* b" *Maple V* folosește notația 'a^b'. Prezența caracterului ';' este obligatorie, făcând parte din sintaxă. După scrierea liniei, aceasta trebuie activată (lansată în execuție) tastând **<ENTER>**. Urmează apoi o atribuire de forma

```
exp2:=solve(nume_exp,x);
```

deoarece expresia o are ca necunoscută pe x.

Rezultatul îl obținem în exp2:=**RootOf**(exx), unde exx va avea forma expresiei nume_exp, necunoscuta x fiind acum înlocuită de _Z. **RootOf** este o scriere reprezentativă pentru rădăcinile ecuației. Vom definitiva acum aflarea rădăcinilor prin

```
allvalues(exp2);
```

Se vor afișa toate rădăcinile ecuației respective. O altă funcție, roots, permite aflarea unor rădăcini ale unei ecuații într-un anumit corp de numere.

Vom da un exemplu de rezolvare al unei ecuații de ordinul patru cu coeficienții foarte mari, problemă care ar putea să apară în calculul polilor unui sistem obținut prin conexiunea în reacție în teoria circuitelor electrice.

În exemplu, după fiecare comandă, este afișat și rezultatul operației respective.

Exemplu

```
> a0:=1234567;
                                   a0:=1234567
> a1:=3456789;
                                   a1:=3456789
> a2:=233445567890;
                                a2:=233445567890
> a3:=1010000040;
                                a3:=1010000040
> a4:=1200040000;
                                a4:=1200040000
> with(GaussInt):
> expres1:=a4*x^4+a3*x^3+a2*x^2+a1*x+a0;
     expres1:=1200040000x^4 + 1010000040x^3 +
      233445567890x^2 + 3456789x + 1234567
> aa:=solve(expres1,x);
   aa:=RootOf(1200040000_Z^4 + 1010000040_Z^3 +
     233445567890_Z^2 + 3456789_Z + 1234567)
> allvalues(aa);
                -.4208119303 - 13.94110428 I
                -.4208119303 + 13.94110428 I
          -.7392404355 10^-5 - .002299652906 I
          -.7392404355 10^-5 + .002299652906 I
```

Caracterul I este folosit pentru a indica partea imaginară, având valoarea I = $\sqrt{-1}$

Domnul prof. drd. Laviniu Bejenaru este cadru didactic la Colegiul Național "Traian" din Drobeta Turnu Severin. Poate fi contactat prin intermediul redacției.

MAPLE V...

- A fost creat de University of Waterloo și Waterloo Maple Software.
- A ajuns la versiunea 6 (Maple V Release 6).
- În funcție de versiune, programul are nevoie de 20 - 60 MB spațiu pe disc și poate rula pe un procesor Pentium, nefiind foarte pretențios din punct de vedere al resurselor de care are nevoie.
- Pentru versiunea 6 sunt necesari 32 MB de memorie RAM.
- Viteza procesorului va afecta rapiditatea calculelor, în cazul ecuațiilor diferențiale (și în general al analizei diferențiale) sau afișărilor grafice de suprafețe date prin funcții în mai multe variabile.
- Pentru mediul științific, o facilitate apărută de la versiunea 3 (Maple V Release 3), este posibilitatea exportării fișierelor în format LaTeX, astfel facilitând scrierea articolelor și cărților care includ aplicații finalizate cu MAPLE.
- MAPLE V este un produs distribuit gratuit care poate fi descărcat de pe Internet de la adresa http://www.maplesoft.com.
- MAPLE V este un produs interactiv și conține un număr mare de rutine compilate care efectuează calcule matematice variate.
- Toate operațiile sunt implementate astfel încât să permită lucrul și cu numere foarte mari.

Războiul PROCESOARELOR

Mihai Scorțaru

Bătălia pentru supremație în industria microprocesoarelor continuă și în acest mileniu. Cei doi combatanți nu au așteptat prea mult pentru a lansa pe piață primele procesoare ale anului 2001. Vom vedea acum cum încearcă Intel și AMD să își asigure un segment cât mai mare de pe piață.

Intel Celeron

Primul procesor al anului 2000 era un *Celeron* cu frecvența de 533 MHz. Și în anul 2001 *Intel* lansează prima un nou procesor; pe 3 ianuarie a apărut primul *Celeron* care are o magistrală cu frecvența de 100 MHz. Factorul de multiplicare al acestui procesor este 8, deci frecvența sa este de 800 MHz.

Prețul unui astfel de procesor este de 170$, pentru cantități ce depășesc 1000 de exemplare. Introducerea acestui procesor va avea ca efect reducerea prețurilor la celelalte procesoare din această familie. Va mai trebui să treacă puțin timp până când noile procesoare *Celeron* se vor impune pe piață. Acest lucru se datorează, în primul rând, faptului că este destul de dificilă realizarea unui overclocking rentabil. Până în prezent s-a reușit forțarea noului *Celeron* 800 astfel încât să lucreze la o frecvență de 936 MHz, dar mult mai ieftinul *Celeron* 600 poate atinge 1008 MHz.

De fapt, un *Celeron* 800 este, în principiu, vechiul *Celeron* 533 cu frecvența magistralei modificată de la 66 la 100 MHz. Unii mai glumeți ar putea spune că tot ce a reușit *Intel* să facă într-un an este modificarea frecvenței unei magistrale.

Prețurile procesoarelor *Celeron* au rămas aproximativ aceleași ca și luna trecută, având o tendință ușor descendentă.

Celeron 533	*$63*	*-$5*
Celeron 566	*$58*	*-$5*
Celeron 600	*$62*	*-$5*
Celeron 633	*$63*	*-$4*
Celeron 667	*$74*	*-$1*
Celeron 700	*$79*	*-$1*
Celeron 733	*$112*	*-$5*
Celeron 766	*$153*	*-$26*
Celeron 800	*$170*	*n/a*

Singura diferență notabilă apare la procesorul *Celeron* 766 al cărui preț a scăzut cu 26$, pentru a deveni mai ieftin decât noul *Celeron* 800.

Intel Pentium III

Singura apariție în această familie de procesoare este un *Pentium III* la 1 GHz cu magistrala de 100 MHz. Introducerea unui astfel de procesor este destul de bizară, mai ales că este și mai scump decât un procesor cu aceeași frecvență, dar cu magistrala de 133 MHz.

Un *Pentium III* 1000/100 costă 469$, în timp ce un *Pentium III* 1000/133 costă 404$. De asemenea, procesoarele *Pentium III* cu magistrala la 100 MHz vor părăsi curând piața, deci acest produs va supraviețui pentru o foarte scurtă perioadă de timp.

Prețurile procesoarelor *Pentium III* cu magistrala de 133 MHz au rămas și ele aproximativ aceleași, păstrând o ușoară tendință descendentă. Prețurile procesoarelor echivalente, cu magistrala de 100 MHz, sunt puțin mai mari.

Pentium III 533	*$114*	*+5$*
Pentium III 600	*$120*	*-14$*
Pentium III 667	*$129*	*-5$*
Pentium III 733	*$139*	*-25$*
Pentium III 800	*$174*	*-5$*
Pentium III 866	*$210*	*-15$*
Pentium III 933	*$289*	*-30$*
Pentium III 1000	*$404*	*-35$*

Datorită faptului că este greu de găsit, prețul procesorului *Pentium III* 533 a crescut puțin. În rest, diferențe notabile de prețuri apar doar la procesoarele cu frecvențe mari. Acest lucru de datorează și faptului că a fost introdus procesorul *Pentium 4* cu frecvența de 1.3 GHz, procesor al cărui preț este accesibil unui număr destul de mare de utilizatori.

Intel Pentium 4

Așa cum am spus, pentru a umple golul între 1 GHz și 1.4 GHz, *Intel* a lansat un procesor *Pentium 4* cu frecvența de 1.3 GHz.

Chiar dacă acest procesor trebuia să apară doar la sfârșitul lunii ianuarie, lansarea sa a fost devansată din motive comerciale.

Astfel, pe data de 3 ianuarie, împreună cu noul *Celeron* 800, *Intel* a anunțat disponibilitatea procesorului *Pentium 4* 1.3 GHz la prețul de 409$. Acest preț este comparabil cu cel al procesorului *Pentium III* 1 GHz, ceea cea arată destul de clar intenția celor de la *Intel* de a încuraja tranziția de la *Pentium III* la *Pentium 4*.

Celelalte procesoare *Pentium 4* disponibile pe piață au frecvențele de 1.4 și 1.5 GHz și costă 591$, respectiv 833$.

AMD Duron

Pentru a-și menține poziția de lideră pe piața procesoarelor destinate calculatoarelor ieftine, *AMD* a lansat oficial pe data de 8 ianuarie procesorul *Duron* 850. Acest procesor era disponibil pe piață încă din decembrie 2000.

Testele comparative între *Celeron* 800 și *Duron* 850 au arătat clar superioritatea procesoarelor de la *AMD*. Acestea sunt mai convenabile și datorită prețurilor mult mai mici la care pot fi achiziționate. În ultima lună, aceste prețuri au mai scăzut puțin.

Duron 600	*$47*	*-$5*
Duron 650	*$41*	*-$5*
Duron 700	*$51*	*-$6*
Duron 750	*$62*	*-$11*
Duron 800	*$73*	*-$20*
Duron 850	*$105*	*n/a*

Introducerea procesorului *Duron* 850 a determinat scăderea semnificativă a prețurilor procesoarelor *Duron* 750 și *Duron* 800, deoarece acestea trebuie să fie mai ieftine decât noul procesor. Un *Duron* 650 este mai ieftin decât un *Duron* 600 datorită faptului că există mai multe astfel de procesoare în stocurile distribuitorilor.

AMD Athlon

Nu există noutăți în această familie de procesoare, *Athlon* 1.2 GHz fiind cel mai rapid procesor de la *AMD*.

Chiar dacă frecvența unui *Pentium 4* 1.5 GHz este cu 25% mai mare decât cea a unui *Athlon* 1.2 GHz, mulți specialiști sunt de părere că, în general, procesorul de la *AMD* se comportă mai bine decât cel de la *Intel*, în aplicațiile folosite de utilizatorul obișnuit.

Totuși, pentru a putea concura cu succes în fața companiei *Intel*, *AMD* a redus semnificativ prețurile acestor procesoare.

Athlon 700	*$86*	*-$1*
Athlon 750	*$90*	*-$6*
Athlon 800	*$101*	*-$29*
Athlon 850	*$102*	*-$37*
Athlon 900	*$130*	*-$35*
Athlon 950	*$156*	*-$62*
Athlon 1000	*$173*	*-$76*
Athlon 1100	*$217*	*-$118*
Athlon 1200	*$262*	*-$221*

În viitor

Pe data de 29 ianuarie 2000 este programată o reducere de prețuri a anumitor procesoare de la *Intel*. Totuși, chiar dacă prețurile acestora vor scădea, ele nu vor ajunge la nivelul celor de la *AMD*. Această a doua companie va oferi în continuare produse mai ieftine și mai performante.

Chiar dacă acum cel mai rapid *Athlon* este mai bun decât cel mai rapid *Pentium 4*, situația nu se va păstra mult timp, deoarece frecvențele procesoarelor *Pentium 4* vor crește mai rapid decât cele ale procesoarelor *Athlon*, iar diferența care va apărea își va spune cuvântul.

Încă așteptăm apariția pe piață a procesoarelor *Athlon* cu magistrala la 266 MHz. Chiar dacă ele au fost lansate în cursul anului 2000 la frecvențe de 1000, 1133 și 1200 MHz, ele nu pot fi încă găsite, datorită lipsei plăcilor de bază.

Mihai Scorțaru este redactor-șef adjunct al GInfo. Poate fi contactat prin e-mail la adresa skortzy@xnet.ro.

Kernelul Linux 2.4

Linus Torwalds, părintele sistemului de operare Linux, a anunțat finalizarea lucrului la versiunea 2.4 a kernelului pentru Linux.

Această versiune, deși a întârziat aproape un an, promite să aibă un rol foarte important pentru comunitatea *open source*.

A fost îmbunătățit suportul pentru multiprocesare și a fost adăugat suport pentru viitorul procesor pe 64 de biți, *Intel Itanium*.

Bucuros că a terminat treaba, Linus Torwalds și-a rugat colaboratorii să nu raporteze bug-uri prea curând pentru că le va ignora.

Structură și stil în PROGRAMARE

Bazil Pârv

Limbajul Pascal a fost creat de Niklaus Wirth în scopuri didactice pentru a oferi celor care învață programare un limbaj care să faciliteze respectarea principiilor programării structurate. Acest limbaj, la ora actuală, este cel mai răspândit limbaj de programare care se regăsește în programele școlare. Cu toate acestea există o serie de cunoștințe fundamentale cărora nu li se acordă suficientă atenție pe parcursul studiului în școli. Mulți elevi, dar și dascălii lor se mulțumesc nu odată, dacă programul "merge" și nu țin cont de eficiență sau de stil. Vom încerca, într-o serie de articole să reliefăm câteva din aceste aspecte.

Structura unui program Pascal

Structura unui program *Pascal* este un lucru binecunoscut, așadar o reamintim doar:

```
Program nume_program; { antetul programului }
  declarații
Begin          { instructiunea compusa (corpul) }
                { programului nume_program }
   instr_1;
   ...
   instr_n
End.      { sfarsitul programului nume_program }
```

Partea de *declarații* poate să fie vidă sau poate să conțină declarații, respectiv definiții de:

- constante (**const**)
- tipuri de date (**type**)
- etichete (**label**)
- subprograme (**procedure** sau **function**)
- variabile globale (**var**)

În condiții *normale* un program este soluția unei probleme. În cele ce urmează vom prezenta câteva tipare de structurare a programelor *Pascal* care, în concepția noastră, ar trebui să respecte următoarele cerințe:

- structura programului principal va fi independentă de problema de rezolvat;
- elementele specifice ale problemei de rezolvat vor fi "ascunse" în subprograme;
- corpul programului principal va conține numai apeluri de subprograme.

O analiză independentă de problema care se rezolvă

Pentru orice problemă *P* pe care dorim s-o rezolvăm, notăm cu *D* mulțimea datelor de intrare și cu *R* mulțimea rezultatelor. Cu aceste notații, specificarea problemei se face astfel:

```
Specificarea P este:
   Date: D
   Rezultate: R
SfSpecificare { P }
```

Pe baza acestei specificări, vom discuta trei modele de structură a algoritmului de rezolvare a problemei, care corespund situațiilor:

- programul prevede execuție pentru un singur set de date;
- programul se va executa pentru mai multe seturi de date;
- execuția poate fi "controlată" cu ajutorul unui meniu.

Execuție cu un singur set de date

Într-o exprimare naturală, rezolvarea problemei *P* se poate descrie algoritmic în felul următor:

```
Algoritmul P este:          { versiunea 0 }
   @Citește datele de intrare D
   @Calculează rezultatele R pe baza datelor de intrare D
```

```
   @Afișează rezultatele R (și eventual datele de intrare D)
SfAlgoritm { P }
```

Descrierea de mai sus nu este completă, deoarece conține numai propoziții nestandard (prefixate de caracterul '@'). Ea exprimă modalitatea de rezolvare a problemei *P* prin descompunerea acesteia în trei subprobleme, fiecare mai simplă decât *P*. Soluția problemei *P* se obține prin combinarea soluțiilor subproblemelor (în acest caz este vorba de compunere secvențială).

Fiecare subproblemă rezultată din descompunere poate fi considerată subalgoritm. Folosim următoarele notații:

- `CitesteDate(DI)` - subalgoritmul corespunzător subproblemei *@Citește datele de intrare* `DI`
- `Calculeaza(DI,RC)` - subalgoritmul corespunzător subproblemei *@Calculează rezultatele* `RC` *pe baza datelor de intrare* `DI`
- `AfiseazaRezultate(DI,RC)` - subalgoritmul corespunzător subproblemei *@Afișează rezultatele* `RC` *(și eventual datele de intrare* `DI`*)*

Cu aceste notații, versiunea `0` a algoritmului *P* devine:

```
Algoritmul P este:                    { versiunea 1 }
  Apelează CitesteDate(D)
  Apelează Calculeaza(D, R)
  Apelează AfiseazaRezultate(D, R)
SfAlgoritm { P }
```

Subalgoritmii de mai sus sunt specificați în cele ce urmează. Singurele elemente care se precizează sunt *precondițiile*, *postcondițiile*, *semnificațiile* și *tipurile parametrilor* (**IN** - de intrare, **OUT** - de ieșire) și scopul subalgoritmului.

```
SubAlgoritmul CitesteDate(DI) este:
  Pre: true
  Post: DI au fost citite
  Parametri:
    DI - datele de intrare ale problemei P (OUT)
  Scop:
    Citește DI de la intrarea standard
SfSubAlgoritm { CitesteDate }

SubAlgoritmul Calculeaza(DI,RC) este:
  Pre: DI au fost citite
  Post: RC au fost calculate
  Parametri:
    DI - datele de intrare ale problemei P (IN)
    RC - rezultatele problemei P (OUT)
  Scop:
    Calculează RC pe baza DI
SfSubAlgoritm { Calculeaza }

SubAlgoritmul AfiseazaRezultate(DI,RC) este:
  Pre: DI au fost citite
       RC au fost calculate
  Post: true
  Parametri:
    DI - datele de intrare ale problemei P (IN)
    RC - rezultatele problemei P (IN)
  Scop:
    Afișează RC (și eventual DI) la ieșirea standard
SfSubAlgoritm { AfiseazaRezultate }
```

Specificarea acestor subalgoritmi conține, pe lângă elementele uzuale, precondițiile și postcondițiile acestora. *Precondiția* este un predicat care specifică condițiile în care algoritmul se poate executa (el exprimă condițiile impuse asupra parametrilor de intrare), iar *postcondiția* este un predicat care este adevărat la terminarea execuției (el exprimă legătura dintre parametrii de intrare și de ieșire). Să remarcăm că ordinea de apelare a subalgoritmilor din versiunea `1` a algoritmului *P* asigură automat verificarea precondițiilor subalgoritmilor respectivi.

Am ajuns astfel la o primă versiune finală a algoritmului *P*, care conține numai apeluri de subalgoritmi. Această versiune este generală, deoarece caracteristicile problemei *P* sunt "ascunse" în:

- datele de intrare `D` și rezultatele `R`, care nu au fost explicitate;
- subalgoritmii `CitesteDate`, `Calculeaza` și `AfiseazaRezultate`, care au fost doar specificați, nu și descriși.

Atât structurile de date (pentru datele de intrare `D` și rezultatele `R`), cât și subalgoritmii enumerați depind de problema *P* care trebuie rezolvată. Analiza lor se amână până în momentul când vom considera un exemplu.

În cele ce urmează vom discuta alte moduri de combinare a soluțiilor subproblemelor pentru a obține versiuni structural diferite ale algoritmului *P*.

Execuție cu mai multe seturi de date

Versiunea `1` a algoritmului *P* corespunde unei execuții cu un singur set de date. Dacă dorim să repetăm execuția cu un alt set de date, programul trebuie lansat din nou în execuție.

Pentru a permite execuția programului cu mai multe seturi de date, următoarele două versiuni ale algoritmului *P* includ cele trei subprobleme (în aceeași ordine) într-o structură repetitivă. Descrierea inițială a algoritmului în acest caz este:

```
Algoritmul P este:                    { versiunea 0R }
  Repetă
    @Citește datele de intrare D
    @Calculează rezultatele R pe baza datelor de intrare D
    @Afișează rezultatele R (și eventual datele de intrare D)
  PânăCând @Nu mai sunt date de intrare
SfAlgoritm { P }
```

Constatăm aici un alt mod de compunere a soluțiilor subproblemelor pentru a rezolva problema inițială: folosirea structurii repetitive **Repetă**...**PânăCând**. De asemenea, să remarcăm că a apărut o subproblemă nouă: *@Nu*

mai sunt date de intrare, care reprezintă criteriul de terminare a execuției programului.

Pentru această subproblemă avem cel puțin două variante de discuție:

- faptul că nu mai sunt date de intrare se stabilește în subproblema *@Citește datele de intrare* D;
- continuarea execuției cu un alt set de date de intrare este separată de citirea datelor de intrare.

Continuarea execuției pe baza datelor de intrare

Versiunea 0R a descrierii algoritmului devine în această situație:

```
Algoritmul P este:                    { versiunea 0R1 }
  Repetă
    @Citește datele de intrare D (dacă mai sunt date de
     intrare)
    dacă @Mai sunt date de intrare
      atunci
        @Calculează rezultatele R pe baza datelor de in-
         trare D
        @Afișează rezultatele R (și eventual datele de
         intrare D)
    SfDacă
  PânăCând @Nu mai sunt date de intrare
SfAlgoritm { P }
```

În această versiune constatăm următoarele:

- s-a modificat subproblema aferentă citirii datelor de intrare (ea trebuie să decidă dacă mai sunt date de citit și să le citească numai în caz afirmativ);
- execuția ultimelor două subprobleme inițiale este condiționată de existența datelor de intrare (specificată prin noua subproblemă *@Mai sunt date de intrare*);
- subproblemele *@Mai sunt date de intrare* și *@Nu mai sunt date de intrare* exprimă situații opuse una celeilalte (când prima este adevărată, a doua este falsă și invers).

Să remarcăm că acum soluția problemei *P* care se obține prin compunerea soluțiilor subproblemelor inițiale, folosește toate structurile de calcul proprii programării structurate: *secvența*, *repetiția* și *decizia*. Operația de compunere a soluțiilor subproblemelor pentru a obține soluția problemei inițiale are acest înțeles: ***secvență*** (soluția unei subprobleme este folosită ca dată de intrare în următoarea problemă ce apare în secvență), ***repetiție*** (grupurile de soluții parțiale se pot repeta), ***decizie*** (o anumită subproblemă se rezolvă numai dacă este îndeplinită o anumită condiție) și ***apel recursiv*** (aceeași subproblemă de rezolvat, cu date de complexitate sau valoare mai mică).

Ținând cont de observațiile de mai sus, descrierea finală a algoritmului *P* este:

```
Algoritmul P este:                    { versiunea 1R1 }
  Repetă
    Apelează CitesteDate2(D,SuntDate)
    Dacă SuntDate
      atunci
        Apelează Calculeaza(D,R)
        Apelează AfiseazaRezultate(D,R)
    SfDacă
  PânăCând not SuntDate
SfAlgoritm { P }
```

Pentru acest algoritm trebuie precizate următoarele:

- subalgoritmii Calculeaza și AfiseazaRezultate au specificarea prezentată anterior în cadrul acestui articol;
- am proiectat un subalgoritm nou: CitesteDate2, corespunzător noii subprobleme de citire a datelor de intrare;
- subproblemele *@Mai sunt date de intrare* și *@Nu mai sunt date de intrare* sunt reprezentate printr-o variabilă locală SuntDate, obținută ca parametru de ieșire din CitesteDate2.

Remarcăm că această variantă de structură a algoritmului folosește numai parțial subalgoritmii prezentați în cazul execuției cu un singur set de date.

Ca observație generală, este bine să respectăm principiul: *orice algoritm rezolvă o subproblemă bine definită.* În cazul nostru, CitesteDate2 are două sarcini: atât decizia privitoare la citirea următorului set de date, cât și citirea propriu-zisă a acestuia. Este de preferat să separăm citirea datelor de decizia de continuare. Acest articol va conține și prezentarea unei variante care realizează acest lucru. Specificarea noului subalgoritm CitesteDate2 este:

```
SubAlgoritmul CitesteDate2(DI,ExistaDate) este:
  Pre: true
  Post: DI au fost citite iar ExistaDate a fost setat
  Parametri:
    DI - datele de intrare ale problemei P (OUT)
    ExistaDate - indicator al existenței datelor citite
                 (OUT)
      ExistaDate = true dacă s-au citit datele de
                        intrare
      ExistaDate = false în caz contrar
  Scop:
    Citește DI de la intrarea standard (dacă există) și
    setează ExistaDate
SfSubAlgoritm { CitesteDate2 }
```

Versiunea 1R1 este o altă versiune finală a algoritmului *P*. Toate observațiile făcute la sfârșitul secțiunii referitoare la algoritmii care se execută pentru un singur set de date sunt valabile și aici: precondițiile subalgoritmilor sunt verificate prin ordinea de apelare a acestora, algoritmul este general, iar toate detaliile referitoare la problema *P* urmează a fi discutate la proiectarea subalgoritmilor.

Continuarea execuției separată de intrare

Această variantă de execuție cu mai multe seturi de date are ca punct de plecare descrierea incompletă 0R2, care este aproape identică cu 0R:

```
Algoritmul P este:                    { versiunea 0R2 }
  Repetă
    @Citește datele de intrare D
    @Calculează rezultatele R pe baza datelor de intrare D
    @Afișează rezultatele R (și eventual datele de intrare D)
  PânăCând not @Se continuă cu un alt set de date
SfAlgoritm { P }
```

În descrierea de mai sus, celor trei subprobleme inițiale li se adaugă o a patra, *@Se continuă cu alt set de date*, căreia îi va corespunde un subalgoritm numit `Continuare`. Varianta finală a descrierii algoritmului *P* este:

```
Algoritmul P este:                    { versiunea 1R2 }
  Repetă
    Apelează CitesteDate2(D,SuntDate)
    Apelează Calculeaza(D,R)
    Apelează AfiseazaRezultate(D,R)
  PânăCând not Continuare
SfAlgoritm { P }
```

Se observă că

- subalgoritmii `CitesteDate`, `Calculeaza` și `AfiseazaRezultate` au specificarea prezentată anterior;
- am proiectat un subalgoritm nou: `Continuare`, corespunzător noii subprobleme de continuare a citirii datelor de intrare.

Subalgoritmul `Continuare` este independent de problema *P*. Deoarece este suficient de simplu, vom preciza direct descrierea sa finală:

```
SubAlgoritmul Continuare este:
  Afișeaza "Continuati (D/N)? ";
  Citește raspuns;
  Dacă raspuns = "D"
    atunci Returneaza true
    altfel Returneaza false
  SfDacă
SfSubAlgoritm { Continuare }
```

Remarcăm că această variantă de rezolvare folosește toți subalgoritmii identificați în secțiunea referitoare la algoritmii executați pentru un singur set de date de intrare, adăugând unul nou, `Continuare`, necesar execuției repetate.

Versiunea `1R2` este a treia versiune finală a algoritmului *P*. Toate observațiile anterioare sunt valabile și aici: precondițiile subalgoritmilor sunt verificate automat prin ordinea în care apar apelurile acestora, algoritmul este general, toate detaliile referitoare la problema *P* urmează a fi discutate la proiectarea subalgoritmilor. În plus, se apelează subalgoritmul `Continuare` care este independent de *P*.

Execuție dirijată de meniu

Variantele de algoritmi prezentate până acum presupun o rezolvare liniară a problemei, în sensul că după ce s-au introdus datele de intrare restul procesului este realizat în secvență, fără intervenția utilizatorului. Mai exact, subalgoritmii `Calculeaza` și `AfiseazaRezultate` se apelează (în această ordine) numai dacă s-au citit datele de intrare, asigurându-ne că precondițiile lor sunt verificate automat.

Ultima variantă a algoritmului, pe care o discutăm aici, este structural diferită de celelalte și dă posibilitatea utilizatorului să comande fiecare pas din execuția programului după dorință, inclusiv terminarea acestuia:

```
Algoritmul P este:                    { versiunea 0M }
  Repetă
    @Afișează meniul și citește operația selectată de către
      utilizator din meniu
    Dacă operație = Citire
      atunci @Citește datele de intrare D
      altfel
        Dacă operație = Calcul
          atunci @Calculează rezultatele R pe baza
                   datelor de intrare D
          altfel
            Dacă operație = Afișare
              atunci @Afișează rezultatele R (și e-
                       ventual datele de intrare D)
            SfDacă
        SfDacă
    SfDacă
  PânăCând operație = Terminare
SfAlgoritm { P }
```

În descrierea de mai sus, celor trei subprobleme inițiale li se adaugă o a patra, *@Afișează meniul și citește operația selectată de către utilizator din meniu*, căreia îi va corespunde un subalgoritm numit `AfiseazaMeniu`. Înlocuind propozițiile nestandard cu apeluri de subalgoritmi obținem varianta următoare a descrierii algoritmului *P*:

```
Algoritmul P este:                    { versiunea 1M }
  Repetă
    Apelează AfiseazaMeniu(operatie)
    Dacă operație = Citire
      atunci
        Apelează CitesteDate(D)
      altfel
        Dacă operație = Calcul
          atunci
            Apelează Calculeaza(D,R)
          altfel
            Dacă operație = Afișare
              atunci
                Apelează AfiseazaRezultate
                                    (D,R)
            SfDacă
        SfDacă
    SfDacă
  PânăCând operație = Terminare
SfAlgoritm { P }
```

Remarcăm că

- subalgoritmii CitesteDate, Calculeaza și AfiseazaRezultate au specificarea prezentată anterior;
- am proiectat un subalgoritm nou: AfiseazaMeniu, corespunzător noii subprobleme de afișare a meniului și de citire a opțiunii utilizatorului.

Subalgoritmul AfiseazaMeniu este independent de problema *P*. El are un parametru de ieșire ce reprezintă codul operației cerute de utilizator. Deoarece este suficient de simplu, vom preciza direct descrierea sa finală:

```
SubAlgoritmul AfiseazaMeniu(operatie) este:
  Repetă
    Afișează "Operatiile posibile sunt:"
    Afișează "1.Citirea datelor de intrare"
    Afișează "2.Calcularea rezultatelor"
    Afișează "3.Afisarea rezultatelor"
    Afișează "0.Terminarea executiei
              programului"
    Afișează "Introduceti optiunea (0-3): "
    Citește cod
  PânăCând cod în {0, 1, 2, 3}
  Selectează cod dintre
    1:operație:=Citire
    2:operație:=Calcul
    3:operație:=Afisare
    0:operație:=Terminare
  SfSelectează
SfSubAlgoritm { AfiseazaMeniu }
```

Remarcăm folosirea structurii de selectare **Selectează...SfSelectează** în locul unor structuri alternative în cascadă.

Examinând atent versiunea 1M a algoritmului problemei *P* constatăm că ea este incorectă, în sensul că nu se mai păstrează ordinea firească de apelare a subalgoritmilor: Calculeaza după CitireDate, respectiv AfisareRezultate după Calculează. Cu alte cuvinte, nu sunt respectate precondițiile algoritmilor respectivi, ei putând fi apelați în orice ordine. Prin urmare, înainte de apelarea lor în algoritmul *P*, trebuie să le verificăm precondițiile. Noua versiune a algoritmului - de data aceasta finală - arată astfel:

```
Algoritmul P este:                { versiunea 2M }
  ExistaDate:=false
  ExistaRezultate:=false
  Repetă
    Apelează AfiseazaMeniu(operatie)
    Selectează operatie dintre
      Citire: Apelează CitesteDate(D)
              ExistaDate := true
              ExistaRezultate := false
      Calcul: Dacă ExistaDate
                atunci
                  Apelează Calculeaza(D,R)
                  ExistaRezultate:=true
                altfel
                  Afișează "Nu s-au citit da-
                            tele de intrare"
              SfDacă
      Afisare: Dacă ExistaRezultate
                atunci
                  Apelează
                    AfiseazaRezultate(D, R)
                altfel
                  Afișează "Nu s-au calcu-
                            lat rezultatele"
              SfDacă
    SfSelectează
  PânăCând operatie = Terminare
SfAlgoritm { P }
```

Am prevăzut doi indicatori de stare pentru verificarea precondițiilor, ExistaDate (pentru Calculeaza) și ExistaRezultate (pentru AfiseazaRezultate). Inițial, aceștia sunt setați pe false și

- la operația Citire (care nu este condiționată) ExistaDate se setează pe true, iar ExistaRezultate pe false;
- la operația Calcul se apelează Calculeaza numai dacă ExistaDate, caz în care se setează ExistaRezultate pe true;
- la operația Afisare se apelează AfisareRezultate numai dacă ExistaRezultate;
- în toate cazurile când precondițiile nu sunt verificate se afișează mesaje de eroare.

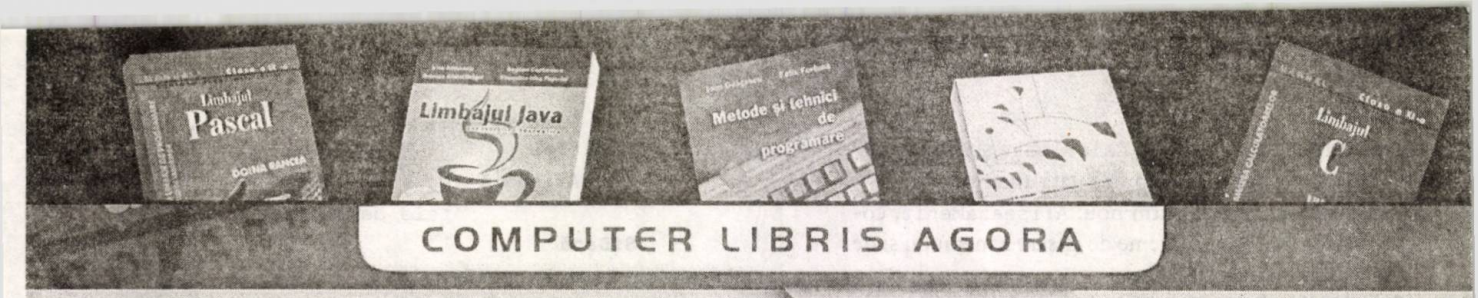
De asemenea, remarcăm înlocuirea structurilor **Dacă ...SfDacă** în cascadă din versiunea 1M cu structura de selecție **Selectează...SfSelectează**, fapt care îmbunătățește exprimarea algoritmului și ușurează înțelegerea sa prin reducerea nivelurilor de indentare.

Versiunea 2M este ultima versiune finală a algoritmului *P*. Aproape toate observațiile făcute la sfârșitul secțiunii referitoare la execuția pentru un singur set de date sunt valabile și aici: algoritmul este general, iar toate detaliile referitoare la problema *P* urmează a fi discutate la proiectarea subalgoritmilor. De data aceasta, precondițiile subalgoritmilor sunt verificate în corpul algoritmului folosind indicatori de stare. În plus, se apelează subalgoritmul AfiseazaMeniu care este independent de *P*.

În numerele următoare

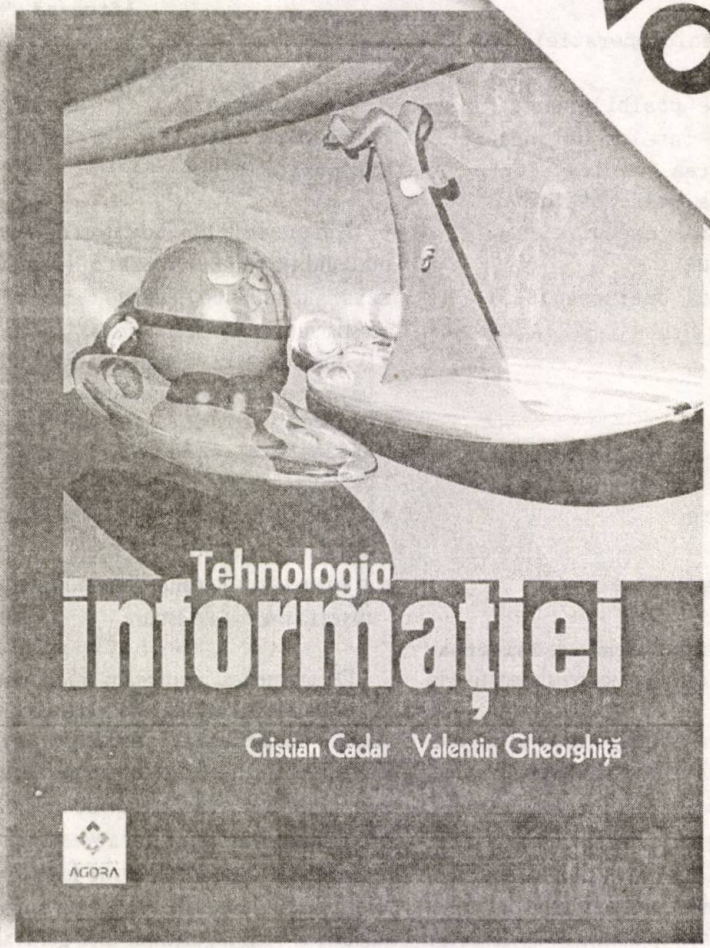
Serialul nostru va continua cu prezentarea modului de proiectare și implementare a algoritmilor generali. Vom prezenta proiectarea tipurilor de date, precum și variante ale algoritmului general de rezolvare a unei probleme. De asemenea vor fi descrise și modurile de implementare a elementelor specifice unei probleme.

Dl. prof. Bazil Pârv este cadru didactic la Universitatea "Babeș-Bolyai" din Cluj. Poate fi contactat prin e-mail la adresa bparv@cs.ubbcluj.ro.



COMPUTER LIBRIS AGORA

NOUTATE



• • Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

AGORA

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Un nou mileniu... cu PROBLEME

Ionuț Stan

Am trăit cu toții, mari și mici, profesori și elevi, bucuria intrării într-un nou mileniu. Visăm la o lume în care să nu avem probleme... Evident nu vorbesc despre "problemele de informatică", de care, cu siguranță, nu vor să se despartă cei pasionați. În cele ce urmează, vă vom prezenta rezolvările a două probleme propuse de către dl. prof. Sorin Groza, apoi propunem spre rezolvare câteva probleme, la care așteptăm rezolvări cât mai reușite.

Serviciul de gardă

În vederea stabilirii serviciului de gardă în noaptea dintre milenii la o unitate militară, comandantul nominalizează *n* soldați (S) și *n* gradați (G). Comandantul, știind că în post intră odată două persoane și că ofițerul responsabil îi pune în pereche pe cei prevăzuți în listă, primul cu ultimul, al doilea cu penultimul etc., stabilește ordinea în listă astfel încât să nu se formeze pereche dintr-un soldat și un gradat. (De ce? Băieții vor înțelege - când vor face armata.) De altfel, comandantul mai știe că *n* este număr par și va folosi această proprietate în momentul în care va întocmi lista.

Scrieți un program care, plecând de la numele și gradele (soldat sau gradat, adică S sau G) celor care vor intra în gardă, afișează pe ecran lista comandantului din care ofițerul va forma perechi, conform cerinței. De asemenea, programul trebuie să afișeze și perechile ce intră în serviciu.

Exemplu

Intrare

```
n=4
Chirila, Macarie, Coza, Domsa, Lador,
Orlea, Hila, Pacurar
S, S, G, G, S, G, S, G
```

Ieșire (o soluție posibilă):

```
lista: Chirila, Coza, Lador, Pacurar,
Domsa, Macarie, Orlea, Hila
serviciul:
Chirila - Hila
Coza - Orlea
Lador - Macarie
Pacurar - Domsa
```

Rezolvare

Numele ostașilor se vor păstra într-un șir de articole numit `ostasi`, fiecare articol având două câmpuri: `nume` și `grad`. Acest șir se va ordona astfel încât în prima parte să apară gradații, apoi să apară soldații. Pentru a crea lista conform cerinței problemei, aceasta se afișează astfel încât să ajungă în pereche fie doi soldați, fie doi gradați. Problema a fost descompusă în subprobleme conform cărora în program s-au implementat subprogramele `Citire_date`, `Ordonare`, `Listare_unul_cate_unul` și `Listare_pe_perechi` ale căror funcție rezultă clar din denumirile lor.

Listing Garda.PAS

```
Program Garda;
uses Crt;
type ostas=record
               nume,grad:string[30]
             end;
     tipostasi=array[1..100] of ostas;
var n,m:Byte;
    ostasi:tipostasi;

procedure Citire_date(var ostasi:
                      tipostasi;var n:Byte);
var i:Byte;
begin
  repeat
    Write('Cati soldati/gradati intra ',
                        'in serviciu? ');
    Readln(n)
  until n mod 2=0;
```

```
  m:=n*2;      { numarul total al celor }
               { care vor efectua garda }
  Writeln('Introduceti datele');
  for i:=1 to m do
  with ostasi[i] do
  begin
    Write('Dati numele ostasului ',i,
                                    ':');
    Readln(nume);
    Write('Dati gradul ostasului ',i,
                        ' (G sau S): ');
    Readln(grad);
    Writeln('--------------------')
  end
end;

procedure Ordonare(var ost:tipostasi;
                                  m:Byte);
var gata:Boolean;
    i:Byte;
    aux:ostas;
begin
  repeat { ordonare cu metoda bulelor }
    gata:=true;
    for i:=1 to m-1 do
              { ordonam sirul de ostasi }
      if ost[i].grad<ost[i+1].grad then
      begin { interschimbam doi ostasi }
        aux:=ost[i];     { atribuire la }
                         { nivel de articol }
        ost[i]:=ost[i+1]; ost[i+1]:=aux;
        gata:=false
      end
  until gata
end;

procedure Listare_unul_cate_unul
                      (ostasi:tipostasi);
            { afisare lista cu ostasi }
var i:Byte;
begin
  Write('lista: ');
  for i:=1 to n div 2 do
    Write(ostasi[i].nume,' ',
                  ostasi[n+i].nume,' ');
  for i:=n div 2 + 1 to n do
    Write(ostasi[n+i].nume,' ',
                 ostasi[i].nume,' ');
  Writeln
end;

procedure Listare_pe_perechi
                      (ostasi:tipostasi);
           { afisare lista cu perechi }
var i:Byte;
```

```
begin
  for i:=1 to n div 2 do
  begin
    WriteLn(ostasi[i].nume,'-',
                    ostasi[n-i+1].nume);
    WriteLn(ostasi[i+n].nume,'-',
                  ostasi[m-i+1].nume)
  end
end;

Begin
  Clrscr;
  Citire_date(ostasi,n);
  Ordonare(ostasi,m);
  Listare_unul_cate_unul(ostasi);
  Listare_pe_perechi(ostasi);
  Readkey
End.
```

Plasarea oglinzilor

Se consideră un șir de n elemente care au valorile 0 și 1. Definim operația plasare a două oglinzi, una la stânga elementului de pe poziția i, iar alta la dreapta elementului de pe poziția j. Această operație are ca efect interschimbarea elementelor egal depărtate de extreme a elementelor cuprinse între pozițiile i și j, inclusiv.

Scrieți un program care va citi două șiruri de lungime n și va stabili pozițiile în care vor trebui plasate cele două oglinzi în cadrul primului șir, astfel încât în urma plasării oglinzilor - conform definiției - primul șir să devină identic cu al doilea șir.

Exemplu

Intrare

```
n = 8
0 0 1 1 0 1 1 0
0 1 0 1 1 0 1 0
```

Ieșire

```
Oglinzile trebuie puse in pozitiile 2 si 6
```

Rezolvare

Pentru a evita toate încercările de a plasa cele două oglinzi, se determină porțiunea maximă de elemente consecutive pe care cele două șiruri nu coincid (nucleul). Astfel se caută cea mai din stânga pereche de elemente ce nu coincid și apoi cea mai din dreapta pereche. Pozițiile corespunzătoare acestor elemente (diferite două câte două) se păstrează în variabilele `a` și `b`. În continuare se verifică dacă oglinzile au efectul așteptat. În acest scop se interschimbă primul element cu ultimul, al doilea cu penultimul etc. din nucleu și se copiază peste vechiul nucleu în oricare dintre cele două șiruri. În urma acestei operații avem un `sir1` nou care se compară element cu element cu `sir2` (neschimbat). Dacă se constată că acum (deci în urma oglin-

dirii) cele două șiruri coincid, se vor afișa pozițiile celor două oglinzi, în caz contrar se va afișa un mesaj corespunzător.

Listing OGLINZI.PAS

```
Program oglinzi
uses Crt;
const mesaj1='Sirurile sunt identice!';
      mesaj2='Nu exista solutie!';
type tip_sir=array[1..100] of Integer;
var nr,     { numar elemente in siruri }
    a, { prima pozitie din primul sir }
       {  unde cele doua siruri nu au }
       {             elemente identice }
    b, { ultima pozitie din al doilea }
       { sir unde cele doua siruri nu }
       {          au elemente identice }
    k,             { indice in nucleu }
    nr_identice,  { numarul elemente- }
            { lor identice din cele   }
            {             doua siruri }
    aux,i:Integer;
    nucleu,       { portiunea unuia din }
       { siruri care trebuie oglindita }
    sir1,sir2:tip_sir;

Begin
  Clrscr;
                              { citire date }
  Write('Numarul elementelor = ');
  ReadLn(nr);
  WriteLn('Introduceti primul sir');
  for i:=1 to nr do Read(sir1[i]);
  WriteLn('Introduceti al doilea sir');
  for i:=1 to nr do Read(sir2[i]);
  { verificarea identitatii sirurilor }
  nr_identice:=0;
  for i:=1 to nr do
    if sir1[i]=sir2[i] then
      Inc(nr_identice);
  if nr_identice=nr then
    Write(mesaj1);
       { determinare pozitii oglinzi }
  if nr_identice<nr then
  begin
    for i:=nr downto 1 do
      if sir1[i]<>sir2[i] then a:=i;
    for i:=1 to nr do
      if sir1[i]<>sir2[i] then b:=i;
                  { extragere nucleu }
    for i:=a to b do
    begin
      Inc(k);
      nucleu[k]:=sir1[i]
    end;
       { efectuare oglindire in nucleu }
    for i:=1 to k div 2 do
    begin
      aux:=nucleu[i];
      nucleu[i]:=nucleu[k-i+1];
      nucleu[k-i+1]:=aux
    end;
       { copierea nucleului oglindit }
    for i:=a to b do
      sir1[i]:=nucleu[i-a+1];
         { verificare efect oglindire }
    nr_identice:=0;
    for i:=1 to nr do
      if sir1[i]=sir2[i] then
        Inc(nr_identice);
    if nr_identice=nr then
      Write('Oglinzile trebuie puse ',
            'in pozitiile ',a,' si ',b)
      else Write(mesaj2)
  end;
  Readkey
End.
```

În continuare vă propunem spre rezolvare câteva probleme simple:

Învățarea alfabetului

Copiii din clasele mici învață din greu alfabetul pe când cei mai mari trag chiulul. În clasa I un copil a "păcătuit" pentru că a încercat să vadă cum este dacă nu își învață "lecția" pentru ora viitoare. Învățătoarea l-a pedepsit punându-l să scrie *n* litere mari, printre care acesta trebuie să scrie cel puțin odată toate literele mari pe care le-a învățat până atunci.

Citind numărul de litere *n* care trebuie să le scrie copilul și literele pe care acestea le-a scris, să se scrie un program care verifică dacă în șirul de litere apar sau nu anumite litere de mai multe ori. Se vor afișa literele care apar de mai multe ori (număr apariții > 1) și numărul aparițiilor acestora.

Exemplu

Intrare

```
5
A B C A C
```

Ieșire

```
A 2
C 2
```

Numere prime... cu divizori

Elevii clasei a V-a au învățat la matematică despre numerele prime că au exact doi divizori: 1 și numărul însuși. La informatică au învățat că numerele se pot reprezenta și în alte baze, nu doar în baza 10. Cei doi profesori s-au gândit să le dea o temă copiilor de clasa a V-a, care să combine ce-

le două informații învățate. Ei vor trebui să găsească toate numerele mai mici decât un număr dat *n*, care au proprietatea că scrise în baza 10 nu sunt prime, dar scrise în altă bază și citite în baza 10, sunt prime.

Scrieți un program care găsește aceste numere, precum și bazele în care au această proprietate.

Exemplu

Intrare

```
n = 5
```

Ieșire

```
numar in baza 10: 4 nu este prim
numar 31 in baza 3 ( in baza 10 este prim)
```

Ghiocei și viorele

Copiii din clasa a VI-a au mers la pădure după ghiocei. Ei au cules ghiocei și viorele, în total un număr de *n* flori de primăvară. Au observat că ghioceii și viorelele erau găsite după o anumită regulă: între două viorele erau ghiocei câte unul, apoi câte doi și tot așa mai departe.

Citind un număr natural *n* care să reprezinte numărul florilor și inițialele lor (G pentru ghiocei, V pentru viorele) să se spună dacă florile se găsesc sau nu într-o configurație care îndeplinește cerința de mai sus. Afișați un mesaj (DA sau NU), precum și pozițiile viorelelor.

Exemple

Intrare

```
n = 6
V G V G G V
```

Ieșire

```
DA
1 3 6
```

Intrare

```
n = 5
G V G G V
```

Ieșire

```
NU
```

Serviciul pe clasă

La începutul anului școlar dirigintele a întocmit planificarea serviciului în clasă, câte doi elevi fiind planificați pentru fiecare zi. Deoarece atunci când au fost întrebați, elevii au declarat că doresc să fie de serviciu câte o fată și un băiat, dirigintele a stabilit următoarea regulă: prima fată din catalog va sta cu primul băiat, a doua cu al doilea și așa mai departe. Dirigintele știe că are *n* elevi în clasă, dar nu a numărat câte fete și câți băieți sunt, așa că se poate întâmpla să constate la sfârșit că unii copii au rămas fără pereche.

Citindu-se numărul de elevi, numele și genul lor (`f` pentru fete, `b` pentru băieți), să se spună dacă au rămas elevi fără pereche și care sunt aceștia.

Exemplu

Intrare

```
n=5
Ana f
Ioana f
Andrei b
Anca f
Paul b
```

Ieșire

```
Fara pereche: Anca
```

Tăierea copacilor

Pe drumul național E81 sunt copaci numerotați în zig-zag (pe partea dreaptă a drumului copacii sunt numerotați cu numere pare și pe stânga cu numere impare). La un moment dat au fost aleși copaci pentru tăiere de pe fiecare parte a drumului după următoarea regulă: se va tăia primul acel copac care este numerotat cu un număr ale cărui cifre însumate formează un pătrat perfect, apoi următorul copac având aceeași proprietate și tot așa mai departe până la capătul drumului.

Citindu-se numărul copacilor, să se determine câți copaci au fost aleși pentru tăiere, pe care parte a drumului sunt aceștia, precum și numărul cu care au fost numerotați. Numerotarea copacilor începe de la 1.

Exemplu

Intrare

```
n = 20
```

Ieșire

```
6 copaci
DE PE DREAPTA: 4 10 18
DE PE STANGA: 1 9 13
```

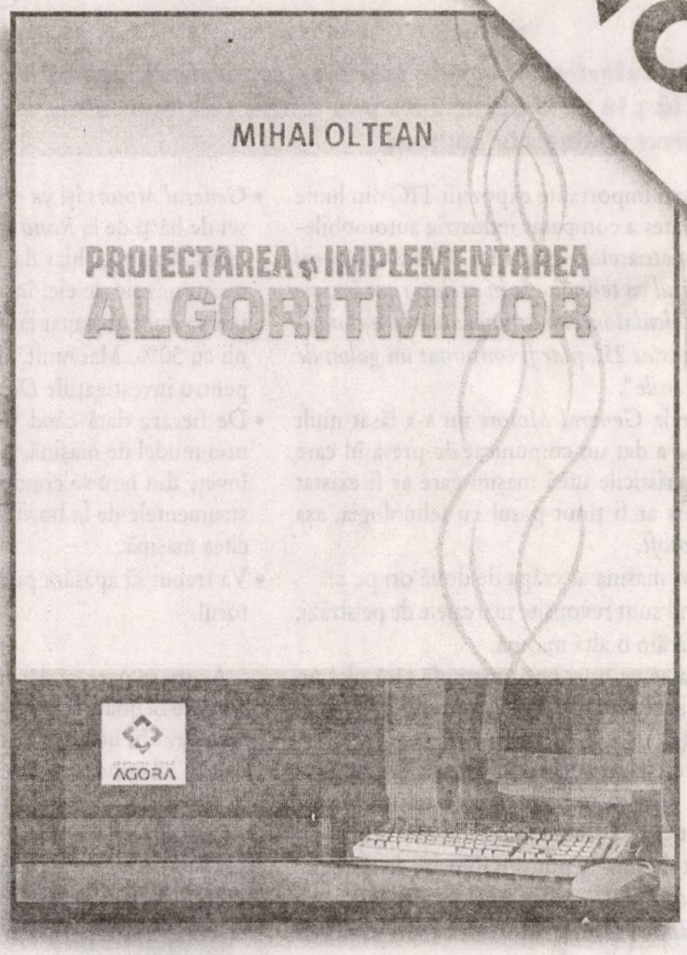
Ionuț Stan este elev la Școala Generală Nr. 1, "Ovidiu Hulea" din Aiud. Poate fi contactat prin e-mail la adresa: aliennt@yahoo.com.

Probleme cu Diablo II Online...

Serverele care găzduiesc varianta online a jocului Diablo II au căzut victime atacurilor unor hackeri care au profitat de o slăbiciune a sistemului de securitate. Hackerii au preluat controlul asupra personajelor diferiților jucători, deseori eliminându-le din joc. Blizzard a înlocuit aceste personaje cu cele salvate pe data de 19 decembrie 2000.

COMPUTER LIBRIS AGORA

NOUTATE



Culegerea constituie un instrumentar pentru cei care doresc să intre în posesia secretelor proiectării algoritmilor. Va fi util când vă pregătiți pentru teză, pentru examenul de bacalaureat sau pentru admitere. Veți găsi în culegere și probleme care pot fi analizate și rezolvate în cercurile de elevi și care vor fi utile în pregătirea pentru concursurile școlare.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

AGORA

GM vs. MICROSOFT

Carmen Şuşman

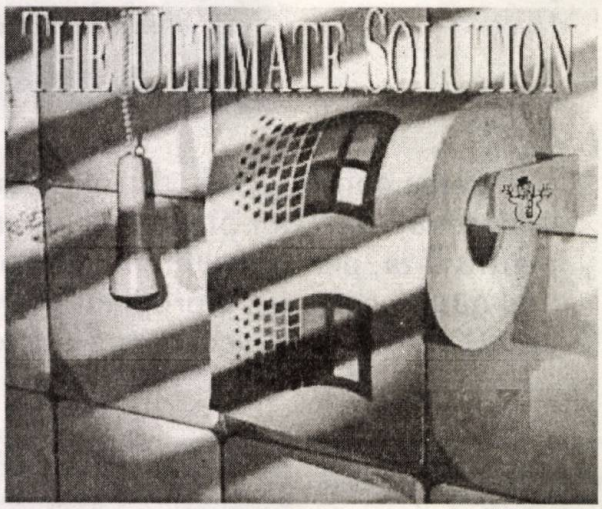
Apar glume referitoare la cele mai diverse domenii, aşa că nici informatica nu a putut fi lăsată de o parte. Iată una dintre cele mai interesante întâmplări care a stârnit zâmbetele multora.

La una dintre cele mai importante expoziţii TIC din lume (COMDEX) Bill Gates a comparat industria automobilelor cu cea a calculatoarelor spunând: "*Dacă General Motors ar fi ţinut pasul cu tehnologia în acelaşi ritm în care a făcut-o industria calculatoarelor, atunci acum am conduce maşini care ar fi costat 25$ şi ar fi consumat un galon de benzină la o mie de mile*".

Replica celor de la *General Motors* nu s-a lăsat mult aşteptată. Compania a dat un comunicat de presă în care erau descrise caracteristicile unei maşini care ar fi existat dacă *General Motors* ar fi ţinut pasul cu tehnologia, aşa cum a făcut-o *Microsoft*.

- Fără nici un motiv, maşina ar crăpa de două ori pe zi.
- De fiecare dată când sunt revopsite marcajele de pe străzi, va trebui să cumpărăm o altă maşină.
- Ocazional, maşina se va opri pe autostradă fără nici un motiv şi noi vom accepta acest lucru, vom reseta şi vom pleca mai departe.
- Ocazional, executarea unei manevre cum ar fi virarea la stânga va duce la oprirea maşinii care va refuza să mai pornească; în acest caz va trebui reinstalat motorul.
- La un moment dat maşina va putea fi folosită de o singură persoană, cu excepţia cazurilor în care este vorba de *Maşina '95* sau *Maşina NT*; dar, în aceste cazuri va trebui să cumpărăm scaune suplimentare.
- *MacIntosh* ar produce o maşină care funcţionează pe baza energiei solare, este fiabilă, de cinci ori mai rapidă, de două ori mai uşor de condus, dar nu va putea fi folosită decât pe 5% dintre străzi.
- Lămpile de avertizare pentru temperatura uleiului, a apei şi pentru alternator ar fi înlocuite cu o singură lampă numită *General Car Default*.
- Noile scaune îi vor forţa pe pasageri să aibă aceleaşi dimensiuni ale spatelui.
- Sistemul de perne de aer va întreba *Sunteţi sigur?* înainte de a se activa.
- Ocazional, fără nici un motiv, maşina ne va închide afară refuzând să ne mai lase înăuntru. În acest caz va trebui ca, simultan, să apăsăm pe clanţa uşii, să învârtim cheia şi să atingem antena radio.
- *General Motors* îşi va obliga toţi clienţii să cumpere şi un set de hărţi de la *Rand McNally* (acum o sucursală a *General Motors*), chiar dacă aceştia nici nu le doresc şi nici nu au nevoie de ele; încercarea dezactivării acestei opţiuni va duce automat la diminuarea performanţelor maşinii cu 50%. Mai mult, *General Motors* va deveni o ţintă pentru investigaţiile *Departamentului de Justiţie*.
- De fiecare dată când *General Motors* va introduce un nou model de maşină, va trebui ca cei care îl cumpără să înveţe din nou să conducă deoarece nici unul dintre instrumentele de la bord nu va funcţiona la fel ca şi la vechea maşină.
- Va trebui să apăsăm pe butonul *Start* pentru a opri motorul.

Acum se pare că *Microsoft* a început să fabrice şi produse care să poată fi folosite exact în scopul căruia i-au fost destinate. Un nou produs de succes al companiei poate fi văzut în imaginea următoare.



Carmen Şuşman este redactor al GInfo. Poate fi contactată prin intermediul redacţiei.

NETREPORT

PCREPORT ÎN MILENIUL III

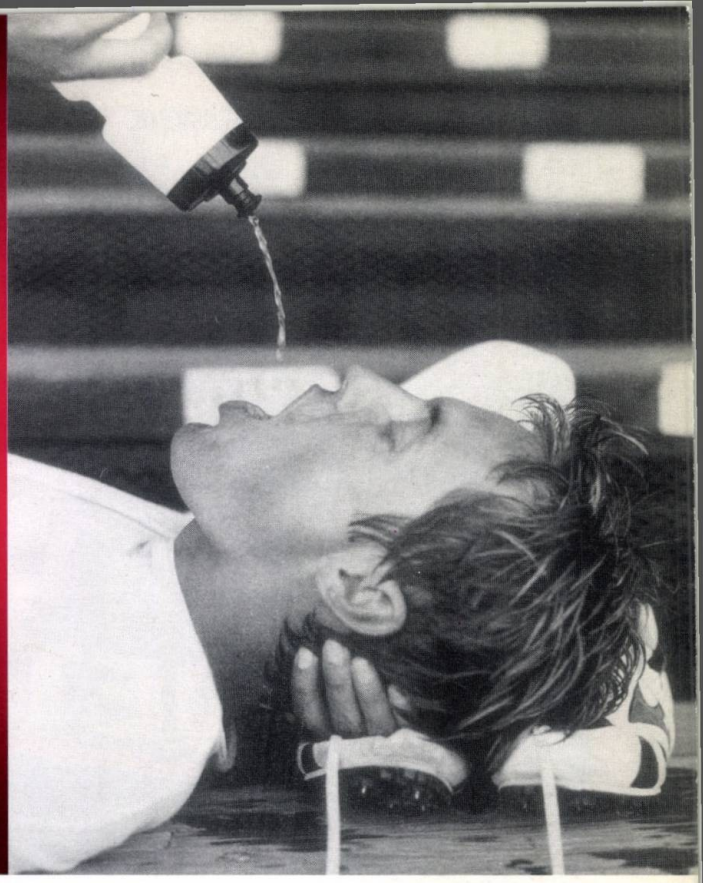
www.pcreport.ro

Conectat la viitorul digital

COMPUTER PRESS AGORA Str. Tudor Vladimirescu nr. 63/1, Tg. Mures, CP 230/11, 4300

Tel: 065-166516, FAX: 065-166290, e-mail: office@agora.ro

RELAXEAZĂ-TE!
E CEL MAI BUN
LUCRU PE CARE ÎL
AI DE FĂCUT.



COMPAQ ARMADA M700

Procesor Intel® Pentium® III 750 MHz • 128 MB SDRAM
• hard disk 12 GB • DVD • placă video AGP 2X • monitor 14" TFT
• modem 56K V.90 +10/100 NIC • Windows 2000 Professional

Restul îl face noul Armada M700 cu procesor Intel®
Pentium® III 750 MHz. Gândit ca tu să nu te mai gândești
la alte probleme. Făcut să le rezolve în locul tău. Să-ți ofere
acces Internet oriunde te-ai afla. Să-ți redea independența
Să câștige timp pentru tine. Să te scape, chiar și pentru puțin
timp, de stresul cotidian. Bun venit în noua eră IT.

COMPAQ
Inspiration Technology

intel inside
pentium® III

DISTRIBUIT PRIN PARTENERII AUTORIZAȚI COMPAQ COMPUTER ROMANIA: MBL Computers - ComputerLand Romă
Tel: (01) 336.07.96; Fax: (01) 336.75.68; **Net Brinel Computers:** Tel: (064) 430.280; Fax: (064) 430.285; **NET Consulti**
Tel: (01) 223.26.80; Fax: (01) 312.41.99; **OMNILOGIC BGS:** Tel: (01) 303.31.00; Fax: (01) 303.31.64; **RomTeamSolutic**
Tel: (01) 311.08.51/311.08.53; Fax: (01) 311.28.30; **SCOP Computers:** Tel: (01) 231.65.78; Fax: (01) 231.65
Sysware Romania S.R.L. : Tel: (01) 313.40.46; Fax: (01) 313.40.44; **Tornado Sistems:** București Tel: (01) 312.75.07; Fax: (01) 312.98
Constanța - Tel: (041) 618.580; Fax: (041) 619.457; **TREND Import Export:** Tel: (01) 211.01.90; Fax: (01) 211.01

Intel, logotipul Intel Inside și Pentium sunt mărci înregistrate ale Intel Corporati

gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

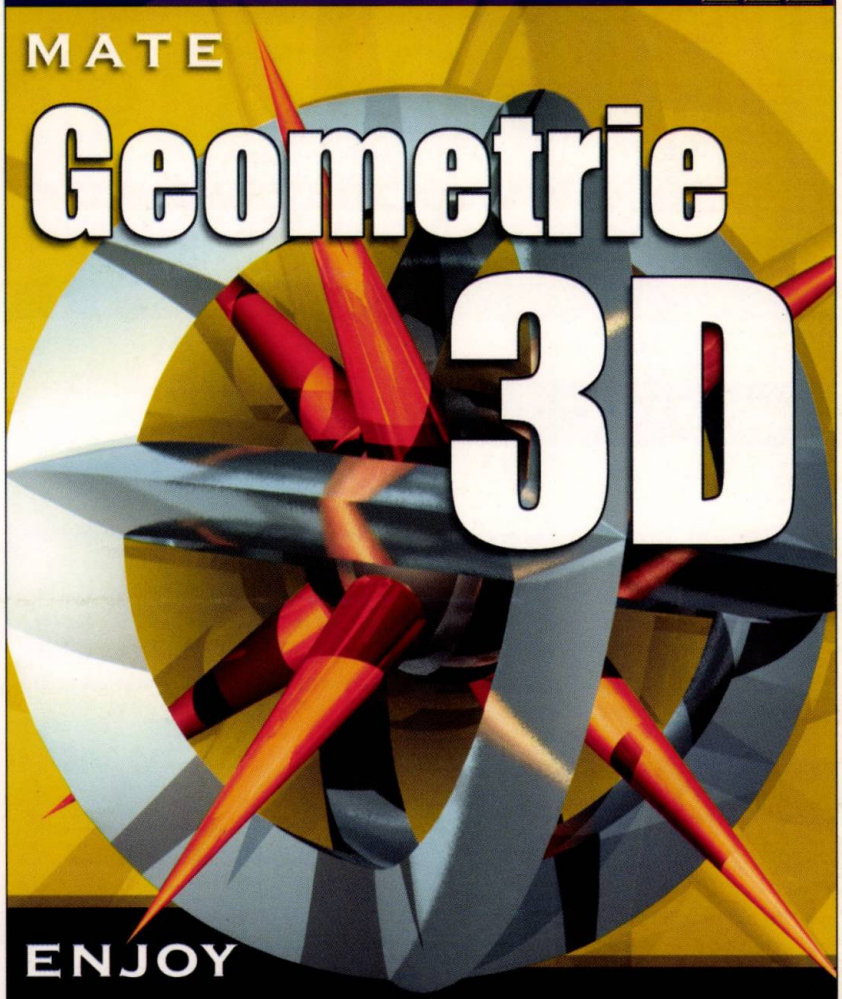
Februarie 2001
Vol. 11/2

Lei 15.000

din sumarul numărului:

MATE

Geometrie 3D



ENJOY

WARCRAFT III

INTERNET Publicarea unui site Web

TEHNO intel contraatacă

intel inside

penti

PC MAGAZINE ROMÂNIA Online

www.pcmagazine.ro

Locul unde în permanență veți găsi ceva ce vă interesează

LABORATOR

www.pcmagazine.ro

Specialiștii laboratorului **PC Magazine România** testează și compară pentru dvs cele mai noi produse existente pe piața românească.
Rezultatele acestor testări pot fi găsite nu numai în paginile revistei, ci și pe sit, unde produsele sunt prezentate pe categorii: **Sisteme**, **Componente, Stocare, Comunicații, Imprimare, Imagine, Divertisment și Software**. Mai mult, pe sit vor fi prezente nu numai produsele testate în ultima lună, ci din ultimul an.

EXCLUSIV PE SIT

www.pcmagazine.ro/exclusiv

EXCLUSIV În capitolul EXCLUSIV sunt publicate articole, interviuri, materiale care completează conținutul numărului curent al revistei.
Până la intrarea la tipografie, în această rubrică erau incluse:

1. Proiectul Legii semnăturii electronice
2. Comunități virtuale - Interviu realizat cu echipa sitului www.necomm.ro
3. Internetul încotro? - Interviu cu Ionuț Tarcea - Marketing Manager Romania Data Systems

ANUNȚURI

www.pcmagazine.ro/anunturi

Anunțuri La cererea cititorilor fideli, a fost inaugurată pagina de anunțuri de vânzări și cumpărări de produse TI&C. Orice vizitator poate trimite anunțuri și le poate citi pe cele deja existente. Anunțurile care nu corespund specificului acestei rubrici nu vor fi afișate.

CONCURS

CONCURSUL LUNII

www.pcmagazine.ro/concurs

În luna martie, sponsorul concursului este firma **Maronet**.

Răspunzând la două întrebări aveți șansa de a câștiga marele premiu: un cont **ACCES INTERNET pentru 1 AN**.

26 de alte premii vă așteaptă: 15 tricouri, 10 conturi a câte 10 ore fiecare, găzduire pagină web și un cont ACCES INTERNET pentru 6 LUNI.

CeBIT în direct

www.pcmagazine.ro/cebit

În perioada târgului de la Hanovra (22-28 martie 2001), ca și anul trecut, corespondenții noștri vor trimite informații fierbinți de la fața locului.
De fapt, veți putea afla informații în avanpremieră despre ce se va petrece la Cebit 2001 începând din 12 martie.

În dialog cu redactorii PC Magazine

Făcând clic pe numele autorului unui articol, îi puteți trimite acestuia opiniile și sugestiile dvs. referitoare la tema și conținutul articolului. Astfel, vom putea ști ce vă interesează și cum putem îmbunătăți conținutul editorial.

Alte servicii

Facilități de care puteți beneficia vizitând situl PC Magazine:
- Arhivă
- Căutare
- Abonamente
- Conținutul CD-ROM-ului

Vol. 11, Nr. 2 /2001

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Redactor:
Carmen Șușman (Alba Iulia)

Au mai colaborat:
Bazil Pârv (Cluj)
Marian Veteanu (Pitești)

Tehnoredactare:
Călin Borz

Coperta și grafica:
Adrian Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca

Adresa e-mail: office@ginfo.ro

Tel: 064/192.422
065/166.516
Fax: 065/166.290

http://www.ginfo.ro

ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359

Editor: Computer Press Agora SRL
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihaela Szabo (mszabo@agora.ro)
Ioana Udroiu (iudroiu@agora.ro)

Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

sumar

actual.. PAG. 5

PROFESORUL nu are întotdeauna DREPTATE

concurs.. PAG. 9

Rezultatele rundei a treia

Runda a cincea

focus.. PAG. 23

HEAP-uri

Mihai SCORȚARU

Secvența de SUMĂ MAXIMĂ

Clara IONESCU

internet.. PAG. 29

Publicarea unui SITE Web

Marian VEȚEANU

Informații utile pentru publicarea unui site pe *Internet*

mate.. PAG. 33

Geometrie 3D

Mihai SCORȚARU

tehno.. PAG. 35

INTEL contraatacă

Mihai SCORȚARU

serial.. PAG. 37

Structură și stil în PROGRAMARE

Bazil PÂRV

Al doilea episod

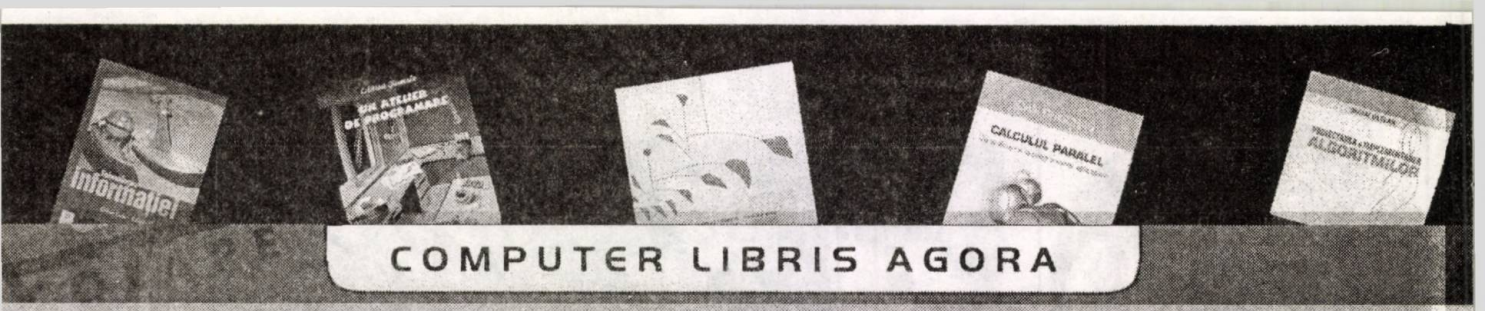
enjoy.. PAG. 42

Sunteți dependenți de CALCULATOR?

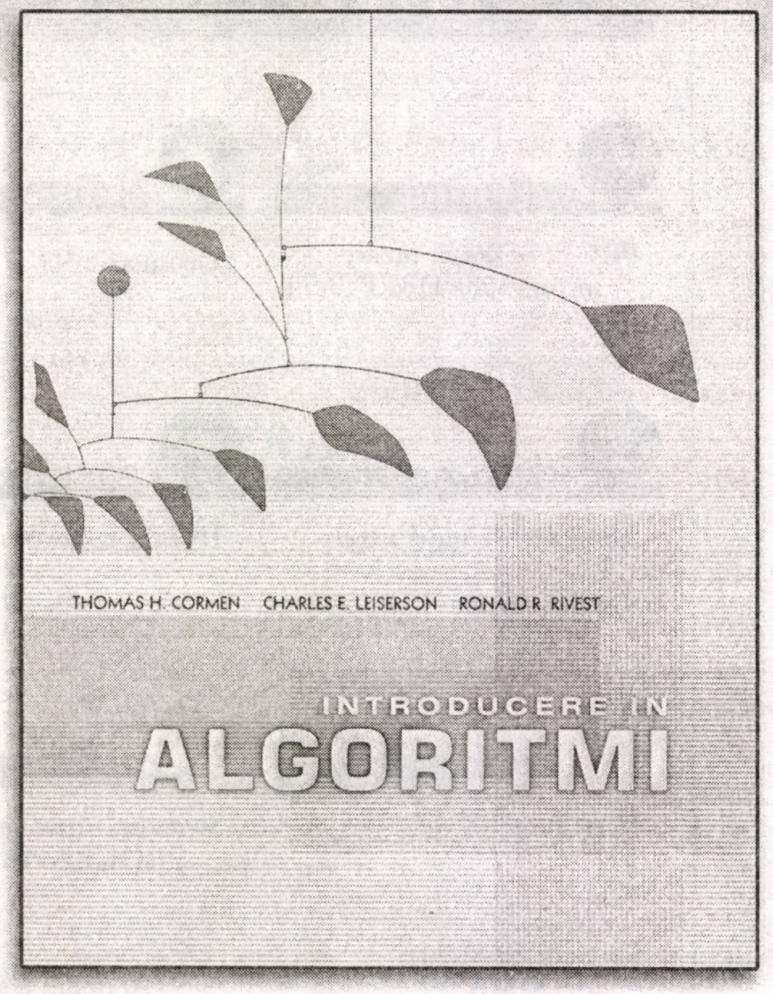
Carmen ȘUȘMAN

Warcraft III

Carmen ȘUȘMAN



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

ELEVI și PROFESORI

Nemulțumiri vor fi pe lumea aceasta cât timp vor exista oameni pe suprafața Pământului. Nemulțumirea constructivă, cea care ne îndeamnă să facem ceva, altfel decât varianta care ne-a nemulțumit, ar trebui să ducă lumea înainte... Dar... Ce facem dacă nu avem mijloace, dacă ne-a pierit graiul? Amintindu-mi de Hemingway, cred și susțin că Omul nu s-a născut pentru a fi învins, el poate fi nimicit, dar învins niciodată...

De unde aceste idei sumbre? De ce nu sunt mai optimistă? După aniversarea a 10 ani de existență a *GInfo* ar trebui să păstrez starea de bucurie și s-o transmit și cititorilor! Da, numai că "realitatea mă minte în față"... Citesc articolul de la pagina 5 și mă întreb: oare câți profesori corectează lucrări de informatică, privindu-le "ca din avion": pe de o parte superficiali, pe de alta plictisiți. Dacă "seamănă" cu ceea ce au predat, e OK, dacă nu, nu poate fi bun!? Sau totuși? Îi dă un 5, că, nu se știe niciodată... Feedback? Discuții pe marginea posibilelor variante, analiza greșelilor frecvente? De unde timp pentru așa ceva? "Bietul" profesor se bucură dacă a reușit să "vizioneze" toate lucrările, să "pună" o notă și să le treacă în catalog. Muncă grea. Mai ales la informatică, unde fiecare scrie ce și cum îl taie capul, nu? Să nici nu mai vorbim despre încurcătura aceasta cu calculatorul. Sărmanii copii și studenți scriu programe pe hârtie, nici ei nu știu dacă au scris un program sau doar un text... Profesorul, în timp record "trebuie" să citească, să compileze în cap, să execute... Dar cum nu are un computer în creier, compilarea mai merge cumva, în schimb execuția o ia razna...

Recent am corectat lucrări de "*Metodica predării informaticii*". Candidații erau profesori care fie predau deja, fie doresc să predea în viitor disciplina informatică în licee. Am pus și eu o întrebare "ciudată": *presupunem că într-o lucrare, un elev a scris că "în antetul unui subprogram trebuie să menționăm cât mai puțini parametri"; stabiliți valoarea de adevăr a propoziției și justificați răspunsul.* Din 70 de candidați 16 nu au răspuns, probabil neștiind dacă afirmația este corectă sau nu. 19 au răspuns corect, menționând printre altele că subprogramele nu trebuie să aibă nici cât mai mulți, nici cât mai puțini parametri, ci exact atâția de câți au nevoie etc. Din păcate, foarte mulți (50%) au răspuns că afirmația este corectă, justificând că toate variabilele se vor declara global pentru a mări lizibilitatea programului, pentru a grăbi execuția etc. Nu mi-am propus să fac expunere despre programare, (o face domnul profesor *Bazil Pârv* cu câteva pagini mai încolo), dar stau și mă întreb: de unde aceste idei? Nu cred că în manuale să fie scris așa ceva. Eu cel puțin, nu am văzut. Atunci? Am bănuiala că stimații noștri colegi, nefiind de fapt programatori, nu s-au izbit de adevăratele cerințe care se pun față de un produs tip program. Ei tind să stabilească reguli noi care le vor ușura munca în momente cum ar fi evaluarea-verificarea sau eventual depanarea unui program.

În timp ce scriu aceste rânduri legate de predarea programării, mă invadează alte amintiri despre alte lucrări în care profesorul descrie o lecție în care se predau cunoștințe legate de un editor de texte, dar lucrarea este plină de greșeli de editare și tehnoredactare. Și să nu amintesc de greșelile de ortografie, care mi se par de neiertat.

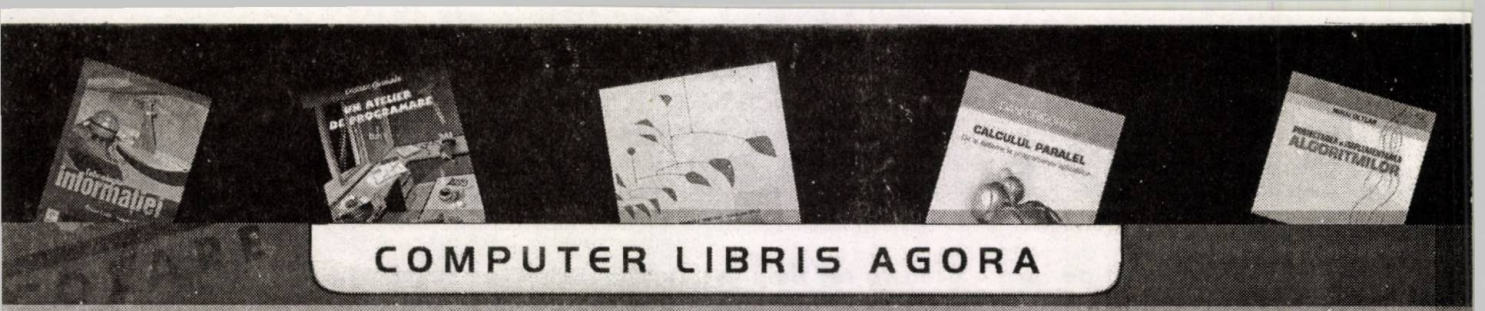
Cine este de vină? Toate acestea sunt problemele mele? Da, sunt ȘI problemele mele, dar mai ales sunt problemele NOASTRE. Cine să ne învețe, cum este bine? Sunt lucruri elementare care nu ridică pretenții intelectuale, nu este vorba de algoritmi pentru olimpici. Dar problemele amintite se propagă până la nivelul olimpicilor și olimpiadelor. Enunțuri ambigue, cu greșeli de ortografie, superficial formulate și redactate. Nu ați văzut așa ceva? În acest caz vă doresc să nici nu vedeți...

Doamna Ministru a Învățământului și Cercetării dă exemple de olimpici excelenți (din păcate doar chimiști) care azi studiază la *MIT*. Oare câți informaticieni li se alătură? De ce credeți că "fug" acești copii până în *America* să învețe în continuare? Sunt multe motive: de la dorința de aventură până la ambiția personală, dar bănuiesc că unul din motive este că acasă nu au fost mulțumiți de "nivel": nivelul predării, nivelul profesorilor etc. Elevul de rând merge la o facultate în speranța că va deveni beneficiarul unor cunoștințe noi, la nivel universitar. Și ce se întâmplă? Sunt două cazuri: nimerește sub aripile unui profesor care le "dă liber" studenților, foști elevi ai claselor de informatică: *"voi nu mai aveți ce învăța, puteți să nu veniți la orele mele"*, sau dimpotrivă unul care se luptă din răsputeri să "repare" ce a intrat în creierele lor cu greșeli (subprograme fără parametri sau instrucțiuni decizionale de forma `if ordonat=true`... etc).

Este nevoie de un pic mai mult respect față de școală, față de ELEV și STUDENT. Cred că de aici ar trebui să pornească totul. Și elevul, și studentul este OM... Sună bizar, dar sunt de-a dreptul îngrozită de câte ori am întâlnit păreri din care rezulta că elevul ar fi doar o ființă obraznică, o nulitate, un escroc, un tembel... (Stop!!!).

Vă doresc și îmi doresc o activitate în care să mergem împreună cu elevii și studenții în căutarea cunoașterii și a cunoștințelor într-un parteneriat echilibrat. Nu uitați, (culmea!) profesorii buni sunt depășiți de unii discipoli de-ai lor. Dacă nu azi, mâine... Așa se măsoară succesul profesorului...

Clara Ionescu



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

PROFESORUL nu are întotdeauna dreptate

Micul Prinț

Se spune că, în România, nu e recomandabil ca un student să își contrazică profesorii pentru că s-ar putea să nu mai termine facultatea. Totuși, există cazuri în care gafele acestora sunt atât de mari încât unii nu se pot abține. Poate că vi se va părea ciudat ca un astfel de articol să fie publicat; motivul principal pentru care am hotărât ca el să apară, este acela de a trage un nou semnal de alarmă - poate că de data aceasta ne va auzi cineva.

Despre UTCN

Cei mai mulți absolvenți de liceu, care doresc să învețe informatică, aleg să urmeze fie o facultate de informatică, fie una de calculatoare. La *Universitatea Tehnică* din *Cluj* există *Facultatea de Automatică și Calculatoare* care este considerată cea mai indicată alegere pentru absolvenții ardeleni pasionați de informatică și care preferă să nu se deplaseze până la *București* pentru a-și continua studiile.

Probabil că pentru a facilita plecarea studenților români în străinătate, *UTCN* a decis să creeze o secție specială care să urmeze cursurile în limba engleză. Chiar dacă, oficial, motivele sunt cu totul altele, foarte mulți studenți se înscriu la această secție, deoarece astfel, după obținerea licenței, le este mult mai ușor să părăsească țara. Chiar în prima zi de facultate are loc un examen de limbă engleză, în urma căruia sunt selectați fericiții care vor studia în engleză.

Bucuria durează destul de puțin deoarece nu trece mult timp până când studenții participă la adevăratele cursuri în engleză. Chiar dacă unii profesori dovedesc o excelentă cunoaștere a acestei limbi, majoritatea lasă de dorit în această privință. Așadar, studenții sunt nevoiți să învețe diferitele variante de engleză corespunzătoare diferitelor materii. Un număr relativ mic de profesori recunosc că nu cunosc foarte bine limba engleză printr-o frază spusă la începutul primului curs. De obicei această frază sună cam așa: "*Trebuie să vă informez că eu nu sunt profesor de engleză ci profesor de... De aceea vă rog să mă scuzați pentru greșelile de limbă pe care le voi face.*" De obicei, aceștia sunt profesorii care vorbesc cel mai bine engleza. Alții vorbesc o romgleză destul de interesantă, în timp ce unii nici nu se mai chinuie și predau în română.

La un moment dat, conducerea *UTCN* a avut ideea ca profesorii care predau la secțiile de engleză să fie supuși unui test. Mai puțin de 25% dintre aceștia au luat "notă de trecere". Replicile celor care nu au trecut testul au fost de tipul: "*Cum puteți spune că eu nu știu engleză? Eu am predat și în Statele Unite și acolo am fost înțeles!*" Este destul de interesantă ideea că a te face înțeles în engleză este echivalent cu a cunoaște această limbă. Nu vom mai insista asupra acestui aspect.

Erorile profesorilor

Se mai întâmplă ca un profesor să greșească dar, în mod normal, acesta ar trebui să își dea seama că a comis o eroare și să recunoască acest lucru. Totuși, așa ceva nu se prea întâmplă la *UTCN*.

De exemplu, un profesor căruia i s-a demonstrat că ceea ce susține nu este adevărat și-a adunat hârtiile și a plecat. Apoi, la examen, mai mult de jumătate dintre studenți au luat note sub 5.

Am mai putea spune că există profesori care predau algoritmică și care susțin că înfășurătoarea convexă a unei mulțimi de puncte nu poate fi determinată în timp polinomial.

Mai există profesori care constată surprinși că studenții rezolvă o problemă mult mai eficient decât credeau ei că este posibil.

Unii profesori sunt atenți la ce spune/scrie studentul și îl felicită dacă acesta are dreptate dar, din nefericire, alții susțin vehement că ideea studentului nu este corectă.

În cele ce urmează vă vom prezenta unul dintre cazurile de acest gen.

Programare funcțională

În anul III studenții de la secția *Calculatoare-Engleză* a *Facultății de Automatică și Calculatoare* din cadrul *Universității Tehnice Cluj* studiază programarea funcțională. La curs sunt predate două limbaje: *RUFL* și *ML*, în timp ce aplicațiile de laborator sunt realizate cu ajutorul limbajului *Lisp*. Acest limbaj este folosit și la seminarii.

La *UTCN*, potrivit regulamentului, datele examinărilor trebuie anunțate. Aceasta nu l-a împiedicat pe un anumit domn asistent să testeze cunoștințele studenților prin surprindere. Acest lucru nu i-a deranjat prea mult pe strudenți, deoarece problemele erau foarte simple. Surpriza a apărut o săptămână mai târziu când s-au comunicat notele obținute.

De exemplu, un student care știa că rezolvase toate problemele a constatat că primise doar un 6. A observat că ultimele două probleme au fost punctate deloc și până la urmă s-a hotărât să întrebe ce a greșit.

Enunțul primei probleme era următorul: *dându-se două mulțimi reprezentate cu ajutorul unor liste, să se determine produsul lor cartezian; rezultatul va fi furnizat sub forma unei liste de perechi.*

Pare destul de simplu, nu? Cei care știu *Lisp* își dau seama că nu e chiar așa. În *Lisp* nu există tablouri și operațiile care ar folosi tablouri în limbajele procedurale trebuie simulate cu ajutorul recursivității. Totuși, aceasta era o problemă foarte simplă, deci nu era prea greu să fie găsită relația recursivă necesară.

Trebuie să mai spunem că în *Lisp*, două operații elementare cu liste constau în extragerea primului element sau extragerea unei liste care conține toate elementele cu excepția primului. Datorită acestui fapt, o relație pentru determinare produsului cartezian ar putea fi următoarea:

$$(\{x\}\cup A)\times(\{y\}\cup B)=\{(x,y)\}\cup(\{x\}\times B)\cup(A\times(\{y\}\cup B))$$

Așadar, funcția recursivă trebuia să aibă ca argumente două liste. În cazul în care unul dintre argumente conține mulțimea vidă, funcția va returna o listă vidă. În caz contrar, funcția va returna o listă care se obține prin concatenarea unei perechi cu alte două liste. Perechea va fi formată din primele elemente ale celor două liste. Prima listă va fi obținută în urma efectuării produsului cartezian dintre mulțimii formată din primul element al primei liste și mulțimea care conține toate elementele celei de-a doua liste cu excepția primului. A doua listă va fi obținută în urma efectuării produsului cartezian dintre mulțimea care conține toate elementele primei liste cu excepția primului și a doua mulțime. Pentru a obține aceste două liste funcția de determinare a produsului cartezian se va autoapela.

Cam aceasta a fost extraordinara idee de rezolvare a acestei probleme. Chiar dacă pe lucrare era scrisă și formula folosită pentru a-l ajuta pe profesor să înțeleagă codul, acesta nu și-a dat seama că rezolvarea era corectă.

Totuși motivul pentru care studentul nu a primit nici un punct a fost formulat astfel. "*Și dacă mă uit din avion îmi dau seama că această rezolvare nu e bună!*". Destul de mirat, studentul a întrebat care sunt motivele pentru care profesorul crede așa ceva. I s-a răspuns: "*Nu cred că problema se poate rezolva cu o singură funcție*".

Studentul a încercat să-l convingă că formula pe care o are în față demonstrează că acest lucru este posibil. Până la urmă a reușit, dar numai într-o anumită măsură. Profesorul a găsit o replică salvatoare: "*Ideea e bună, dar sigur ai greșit la implementare!*". Nu putea să îl contrazică deoarece, de această dată, avea dreptate. Totuși, mulți alți studenți s-au încurcat în paranteze (cei care cunosc limbajul știu că nu e prea greu) sau au avut alte greșeli minore, dar totuși au primit 75% din punctaj. La un moment dat, un student s-a plâns că se pierde în aceste paranteze și i s-a răspuns: "*Parantezele sunt singura chestie logică din Lisp*". Nu credem că are rost să comentăm această afirmație.

Studentul i-a cerut apoi profesorului să îi explice ce nu e corect la cea de-a doua problemă.

Enunțul acesteia era următorul: *dându-se o mulțime reprezentată sub forma unei liste, să se determine mulțimea tuturor submulțimilor acestei mulțimi; rezultatul va apărea sub forma unei liste de liste; mulțimea vidă este reprezentată printr-o listă vidă.*

Din aceleași motive pe care le-am amintit la prima problemă, trebuia găsit un algoritm recursiv. Se știe că primul element al mulțimii apare în jumătate din submulțimi și nu apare în cealaltă jumătate. Pentru a găsi soluțiile se va folosi un așa-numit **parametru de acumulare**. Funcția va avea ca parametri mulțimea ale cărei submulțimi trebuie determinate și parametrul de acumulare. Dacă mulțimea este vidă, se va returna valoarea din parametrul de acumulare. În caz contrar, se va autoapela funcția pentru a determina submulțimile mulțimii din care se elimină primul element. Vor exista două autoapeluri: pentru primul se va adăuga primul element al listei la parametrul de acumulare, iar pentru al doilea nu se va adăuga.

Aceasta a fost cea de-a doua idee genială a studentului. Se pare că nici de această dată nu a fost înțeles. Totuși, nu i s-a mai spus că se vede din avion că rezolvarea nu este corectă, ci că nu a rezolvat problema așa cum a fost ea prezentată în timpul orelor de laborator. La fel ca și în cazul primei probleme algoritmul era descris pe lucrare. Mai mult, studentul a desenat întreg arborele de recursivitate pentru o mulțime cu trei elemente. Pentru a-i demonstra că are dreptate, studentul a început să îi explice algoritmul, exemplificând pe arborele care îl avea în față.

Se pare că profesorul s-a cam plictisit, așa că a spus: "*Uite, facem un pariu. Mergi la un calculator, implementezi algoritmii tăi și dacă funcționează corect îți dau punctele pentru probleme. Te las chiar să schimbi unele lucruri, dar scheletul să rămână același. Dacă tu rezolvi problemele în patru rânduri, atunci recunosc că ai dreptate.*" Studentul nostru nu avea nimic de pierdut așa că a implementat algoritmii. Până la urmă a reușit să îi facă să funcționeze. Ambele funcții aveau patru rânduri.

Era prea evident că studentul avea dreptate, așa că nota i-a fost modificată, dar punem și noi două întrebări:

1. Chiar se justifică o aemenea luptă pentru a convinge un profesor că răspunsul studentului este corect?
2. Oare câți studenți au preferat să tacă sau și-au aplecat capul rușinați?

Ziua examenului

Examenul final a constat din trei subiecte. Primele două erau subiecte de teorie și trebuia folosite limbajele *RUFL* și *ML*. Al treilea a fost un subiect de *Lisp* și a fost o surpriză oarecum plăcută pentru toți studenții.

La primul subpunct li se cerea să scrie o funcție pentru implementarea produsului cartezian. Datorită experienței dobândite, studentul nostru a considerat că este mai indicat să implementez un algoritm care să folosească două funcții. Totuși, a prezentat și varianta cu o singură funcție.

La al doilea subpunct se cerea rezolvarea unei probleme care, până la urmă, se putea reduce la determinarea produsului cartezian a trei mulțimi. Probabil că domnul profesor se aștepta să vadă dacă apare și de această dată o versiune cu o singură funcție. Dar, nu s-a mai întâmplat așa; probabil că studentul nu a mai vrut să riște. Se pare că decizia sa a fost corectă deoarece a primit nota 10 pentru rezolvările sale.

Totuși, chiar și pentru această problemă putem găsi o formulă care poate fi folosită la o implementare cu o singură funcție:

$$(\{x\}\cup A)\times(\{y\}\cup B)\times(\{z\}\cup C)=\{(x,y,z)\}\cup(\{x\}\times\{y\}\times C)\cup$$
$$\cup(\{x\}\times B\times(\{z\}\cup C))\cup(A\times(\{y\}\cup B)\times(\{z\}\cup C))$$

Un alt fenomen interesant care trebuie amintit este acela că au fost și alți studenți care, la examenul de laborator, au rezolvat problema produsului cartezian așa cum am descris anterior. Surprinzător este că ceilalți studenți au primit punctele corespunzătoare rezolvării.

Singura diferență era aceea că rezolvările celorlalți aveau mai mult de patru rânduri. Dar cum lucrările au fost corectate "din avion" se pare că cei care scriu cu "fonturi" mai mari, au avut noroc.

Concluzie

Nu putem contesta cunoștințele de *Lisp* ale profesorului respectiv. Totuși, credem că îi lipsesc unele noțiuni de algoritmică pe care un absolvent de facultate ar trebui să le cunoască. Dacă ne gândim puțin, acest lucru nu ar trebui să ne mai mire. Cei care posedă cunoștințe solide de informatică preferă să plece în străinătate. Dintre cei care aleg să rămână în *România*, majoritatea lucrează la diferite firme și nu aleg o carieră universitară datorită salariului foarte mic. Nu este sarcina noastră să găsim o rezolvare a acestei probleme, și chiar dacă am găsi una probabil că nu ar fi luată în seamă. Așadar, urăm succes noului guvern în rezolvarea problemelor din învățământul românesc.

Încheiem aici, exprimându-ne speranța că cine trebuia să audă a auzit. Din păcate acest caz nu este unul singular. Mulți studenți se plâng de modul în care sunt tratați de unii profesori "atotștiutori". Majoritatea acestora nu se pot face auziți.

Știm că unor profesori nu le va face plăcere dacă vor auzi de publicarea acestui articol. Nu am vrut să jignim pe nimeni, dar cineva trebuia să încerce să tragă semnalul de alarmă. Nu ne așteptăm să rezolvăm ceva; probabil că puțini dintre profesorii de la *UTCN* citesc *GInfo*; majoritatea cadrelor didactice de la *Automatizări și Calculatoare* nici măcar nu știu de existența acestei reviste.

Urăm succes tuturor studenților care încearcă să prezinte la examene soluții diferite de cele ale profesorilor. Celor care preferă să învețe pe de rost ceea ce predau profesorii, nu putem decât să le spunem că, după terminarea facultății, nu va mai sta lângă ei nici un profesor care să îi învețe cum se rezolvă o problemă. Oricum, nu rezolvarea *unei* probleme trebuie învățată, ci ***modul în care se rezolvă probleme, în general.***

Pentru alți oropsiți ai soartei...

Așteptăm cu interes scrisori care să prezinte întâmplări neplăcute (dar și plăcute!) pe care le-au trăit elevi sau studenți din întreaga țară. Din motive lesne de înțeles, nu vom publica numele celor care ne vor scrie decât dacă își vor da acordul. Poate că astfel vom reuși să rezolvăm unele dintre problemele cu care se confruntă aceștia.

Așa cum este normal, vom acorda drept la replică celor care "se simt cu musca pe căciulă".

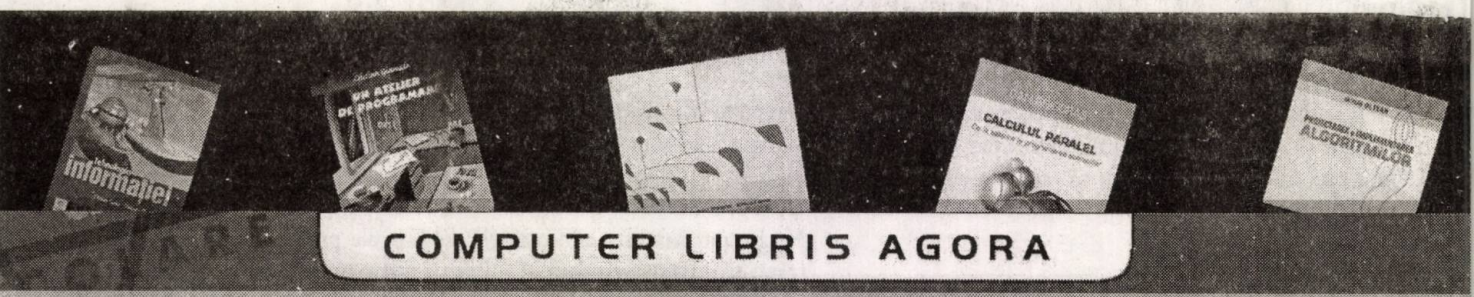
Vă recomandăm să fiți atenți deoarece, în România, încă sunt valabile următoarele reguli:

1. Profesorul are întotdeauna dreptate!
2. Dacă profesorul nu are dreptate, atunci se aplică regula 1.

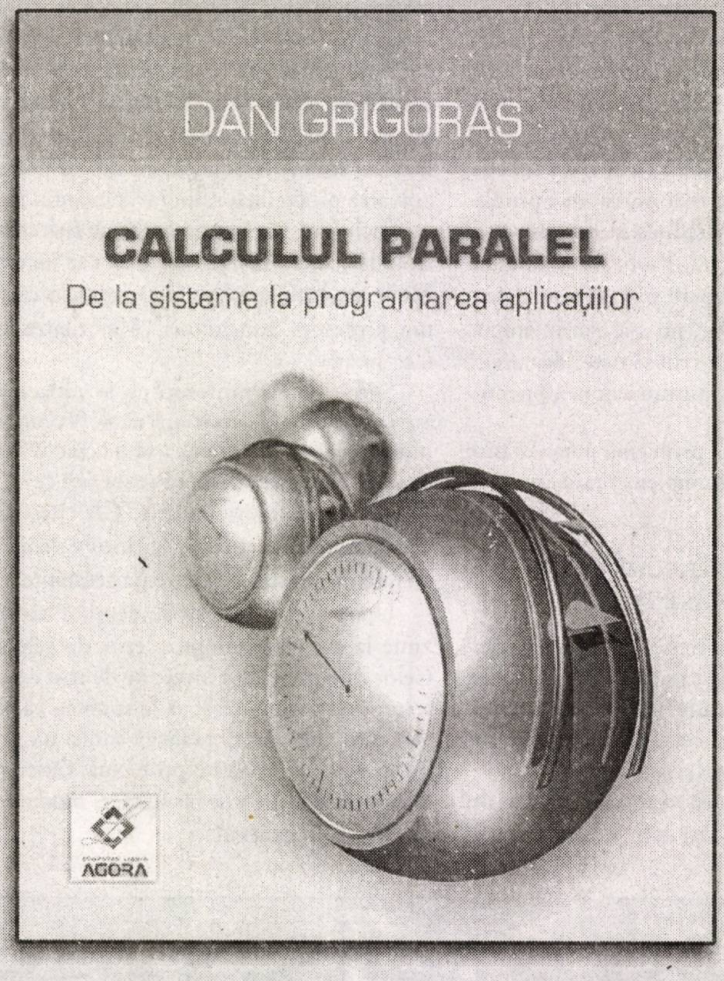
Redacția GInfo nu poate fi făcută responsabilă pentru evenimentele neplăcute care ar putea apărea în urma publicării unei astfel de scrisori. Întreaga responsabilitate cu privire la conținutul scrisorilor revine autorilor!

Noi încercăm doar să vă punem la dispoziție o modalitate prin care să vă puteți exprima punctul de vedere.

De asemenea, așteptăm opiniile tuturor celor care vor să își exprime un punct de vedere referitor la această problemă!



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

CONCURSUL de programare AGORA

Cea de-a treia rundă a Concursului de Programare Agora s-a încheiat. În acest număr vă vom prezenta soluțiile oficiale ale celor trei probleme, precum și rezultatele obținute de concurenți la această etapă. De asemenea, veți putea vedea care este clasamentul intermediar, după trei etape.

Cele trei probleme de la a treia rundă au fost mult mai ușoare decât cele de la a doua. Dacă la runda anterioară doar trei concurenți au obținut punctajul maxim, de data aceasta avem 22 de participanți cu 300 de puncte.

Majoritatea participanților cunoșteau deja soluțiile celor trei probleme. Prima era o generalizare a unei probleme propuse spre rezolvare la concursul Bursele Agora de anul trecut. O rezolvare pentru a doua problemă a fost publicată în numărul 2/1994 al *Gazetei de Informatică*. O indicație pentru rezolvarea optimă a problemei putea fi găsită în cartea *Introducere în algoritmi*, scrisă de *Thomas H. Cormen, Charles E. Leiserson* și *Ronald R. Rivest*, a cărei traducere în limba română a apărut la editura *Computer LIBRIS Agora* din Cluj. Soluția pentru cea de-a treia problemă putea fi găsită în cartea *Proiectarea și implementarea algoritmilor*, scrisă de *Mihai Oltean* și publicată la aceeași editură.

A existat și la această rundă o problemă pentru rezolvarea căreia era nevoie de simularea operațiilor cu numere întregi mari, dar de această dată numărul de cifre era relativ mic.

Nici la această rundă nu am scăpat de unele erori datorate neglijenței. Au fost concurenți care au citit datele de intrare din fișierul `GHOST.IN` în loc de `GHOSTS.IN`, din `UNITS.IN` în loc de `UNIT.IN` sau din `GALBENI.IN` în loc de `GOLD.IN`. Pentru a corecta eroarea ar fi trebuit să modificăm sursele; am preferat să nu facem acest lucru.

De asemenea au mai fost cazuri în care structura arhivei nu a respectat cerințele din regulament sau cazuri în care fișierul `RUNDA3.BAT` conținea comenzi care cereau compilarea altor fișiere decât cele care ar fi trebuit. Aceste erori erau mai ușor de corectat, așadar cei care au comis astfel de erori au fost "iertați".

Se pare că există încă participanți care nu înțeleg că folosirea unit-ului `Crt` duce la apariția unei erori de execuție dacă programul este rulat pe un calculator dotat cu procesor *Intel Pentium III*. Corectarea acestei erori ar fi implicat fie modificarea surselor, fie folosirea unui *patch* care corectează această eroare din *Borland Pascal 7.0*. Din nou am considerat că cea mai bună soluție este să îi "penalizăm" pe concurenți.

Atunci când am publicat regulamentul, am precizat că folosirea unit-ului `Crt` va duce la erori de execuție, dar nu am interzis folosirea lui. Am considerat că este suficient să reamintim participanților că programele lor nu vor funcționa corect dacă acest unit este folosit. Se pare că nu a fost suficient, așadar suntem nevoiți să modificăm acum regulamentul:

> **Începând cu cea de-a cincea rundă este interzisă folosirea unit-ului `Crt` de către cei care folosesc limbajul *Pascal* pentru rezolvarea problemelor. Concurenții care vor include secvența `uses Crt` în programele lor vor fi descalificați!**

Probabil mulți vor considera că descalificarea este o măsură mult prea dură, dar poate așa cei care au greșit vor mai verifica o dată înainte de a trimite o soluție dacă programul folosește unit-ul `Crt`.

Pentru a evita astfel de probleme, redacția GInfo recomandă folosirea limbajului C++ pentru rezolvarea problemelor propuse la *Concursul de Programare Agora*!

Mult succes tuturor participanților!

RUNDA 3 - Top 50

Marius Andrei, Bucureşti	**300**	**George Drumea, Bucureşti**	**300**
Mugurel Andreica, Bucureşti	**300**	**Ioana Ileană, Alba Iulia**	**300**
Liviu Andreicuţ, Bucureşti	**300**	**Liviu Lalescu, Craiova**	**300**
Cristian Băicoianu, Ploieşti	**300**	**Maximilian Machedon, Bucureşti**	**300**
Ciprian Baicu, Dr. Tr. Severin	**300**	**Silvestru Negruseri, Bistriţa**	**300**
Valentin Bişa, Lugoj	**300**	**Mihai Pantiş, Cluj-Napoca**	**300**
Denis Bogdănaş, Iaşi	**300**	**Mihai Pătraşcu, Craiova**	**300**
Ciprian Cană, Vaslui	**300**	**Bogdan Stan, Câmpina**	**300**
Adrian Cârcu, Bistriţa	**300**	**Mihai Stroe, Bucureşti**	**300**
Daniel Crişan, Rm. Vâlcea	**300**	**Radu Ştefan, Braşov**	**300**
Andrei David, Rm. Vâlcea	**300**	**Radu Vătavu, Suceava**	**300**

23. Victor Asavei, Dr. Tr. Severin	297	35. Liviu Păunescu, Constanţa	203
23. Bogdan Nicolae, Sibiu	297	35. Cristian Toth, Lugoj	203
25. Gabriel Pârvan, Râmnicu Sărat	293	39. Livia Bana, Dr. Tr. Severin	200
26. Csaba Andras, Oradea	279	39. Laurent Demonet, Franţa	200
26. Dan Ghinea, Bucureşti	279	39. Vincent Groenhuis, Olanda	200
28. Horea Coroiu, Cluj-Napoca	263	39. Claudiu Gruia, Bucureşti	200
29. Radu Dondera, Bucureşti	250	39. Bogdan Lucaciu, Dr. Tr. Severin	200
30. Codruţ-Lucian Lazăr, Câmpeni	218	39. Mihail Popa, Bucureşti	200
31. Bogdan Piloga, Timişoara	216	39. M. A. Safari Ghahsareh, Iran	200
32. Victor Vernescu, Constanţa	207	39. Vlad Vâlceanu, Arad	200
33. Sorin Otescu, Braşov	206	47. Andrei Markovits, Satu Mare	196
33. Dan Popovici, Arad	206	48. Vencel Bors, Oradea	191
35. Vlad Dascălu, Bacău	203	48. Andrei Csibi, Cluj-Napoca	191
35. Daniel Dumitran, Bucureşti	203	50. Cristian Alexandrescu, Botoşani	190

Runda 3, problema 1

Galbeni

Această problemă a fost rezolvată corect de către 53 dintre participanții la cea de-a treia rundă a ***Concursului de Programare Agora***. Varianta cu trei grămezi de galbeni era destul de cunoscută, rezolvarea acesteia fiind prezentată în numărul 10/5 (mai 2000) al *GInfo*.

Putem considera primele trei grămezi și folosi algoritmul pentru a rămâne cu două grămezi. La aceste două grămezi vom adăuga cea de-a patra grămadă și apoi vom aplica același algoritm. Astfel, din primele patru grămezi am rămas cu două. La acestea vom adăuga cea de-a cincea grămadă și vom folosi același algoritm. Continuând în același mod, vom ajunge la ultima grămadă și vom rămâne, în final, cu numai două grămezi.

În continuare vom prezenta soluția problemei cu trei grămezi. Să presupunem că, la un moment dat, numărul galbenilor din cele trei grămezi este a, b și c. Vom încerca să determinăm o secvență de operații astfel încât să obținem în cele trei grămezi a', b' și c' galbeni cu proprietatea $min(a', b', c') < min(a, b, c)$. Aplicând de mai multe ori această operație vom ajunge în situația în care minimul va ajunge la 0, deci vom rămâne cu numai două grămezi. Deoarece la fiecare pas minimul va scădea, numărul de galbeni dintr-o grămadă va ajunge după un număr finit de pași la 0.

Să presupunem că ne aflăm în configurația (a, b, c). Fără a restrânge generalitatea, putem presupune că $a \le b \le c$. (Dacă nu ne aflăm în această situație putem să permutăm grămezile.)

Vom trasforma această configurație în (a', b', c') astfel încât $b' = rest[b/a]$. Este evident că b' este mai mic decât a (fiind restul unei împărțiri la a), deci $b' < min(a, b, c)$ adică, $min(a', b', c') < min(a, b, c)$.

Pentru a realiza această transformare vom calcula $x = [b/a]$. Analizăm biții corespunzători reprezentării binare a lui x. Fiecare bit care are valoarea 1 poate fi "anulat", transformând configurația (a, b, c) în $(2 \cdot a, b - a, c)$. Pentru un bit care are valoarea 0 nu vom mai modifica valoarea b, ci va trebui doar să deplasăm cu o poziția la stânga biții din reprezentarea binară a lui a. Acest lucru se realizează prin transformarea configurației a, b, c în configurația $(2 \cdot a, b, c - a)$. După parcurgerea tuturor biților lui x vom obține în cea de-a doua grămadă $rest[b/a]$ galbeni.

Vom demonstra în continuare că nu vom avea niciodată un număr negativ de galbeni în nici una dintre cele trei grămezi.

Este evident că numărul elementelor din prima grămadă se dublează la fiecare pas, deci aceasta nu va conține niciodată un număr negativ de galbeni.

Numărul elementelor din a doua grămadă scade permanent, dar în final are valoarea $rest[b/a]$, deci este un număr pozitiv. Ca urmare, nici ce-a de-a doua grămadă nu va conține un număr negativ de galbeni.

Numărul elementelor din cea de-a treia grămadă scade și el permanent. Vom calcula acum valoarea sa minimă. Este evident că avem $a + b + c = a' + b' + c'$. Mai știm că $b' = rest[b/a]$, deci avem $a + b' + a \cdot x + c = a' + b' + c'$, unde $x = [b/a]$. Scăzând valoarea b' din ambii membri obținem $a \cdot (1 + x) + c = a' + c'$. Pentru fiecare bit al lui x, valoarea a se dublează. Având în vedere că numărul biților lui x este $[\log_2 x] + 1$, a' va avea cel mult valoarea $2 \cdot x \cdot a$. Ajungem astfel la inegalitatea $a \cdot (x + 1) + c \le 2 \cdot x \cdot a + c'$ sau $c' \ge c - a \cdot (x - 1)$. Știm că $x = [b/a]$, deci $b \ge a \cdot x$. Deoarece $c \ge b$ avem $c \ge a \cdot x$. Astfel inegalitatea devine $c' \ge a$. Deoarece valoarea a este pozitivă, rezultă că și valoarea c' este pozitivă.

Deoarece numărul de elemente din nici una dintre cele trei grămezi nu poate ajunge negativ, este clar că algoritmul este corect și poate fi aplicat pentru a rezolva problema în varianta cu trei grămezi.

Vom porni cu primele trei grămezi și apoi, pe măsură ce numărul de elemente dintr-o grămadă va deveni zero, vom adăuga celelalte grămezi. Versiunea oficială a soluției acestei probleme este prezentată în continuare.

Listing GOLD.CPP

```
#include <stdio.h>

long gold[1000],n;
FILE *f;

void ReadData(void){
  f=fopen("GOLD.IN","rt");
  fscanf(f,"%ld",&n);
  for (int i=0;i<n;i++)
    fscanf(f,"%ld",&gold[i]);
}

void ThreePiles(int& a,int& b,int& c){
  int ia,ib,ic;
```

```
  long *pa,*pb,*pc;
  while (1){
    if (gold[a]>gold[b]){
      pa=&gold[ia=b];
      pb=&gold[ib=a];
    }
    else{
      pa=&gold[ia=a];
      pb=&gold[ib=b];
    }
    if (gold[c]>=*pb)
      pc=&gold[ic=c];
    else{
      pc=pb; ic=ib;
      if (gold[c]>=*pa)
        pb=&gold[ib=c];
      else{
        pb=pa; ib=ia;
        pa=&gold[ia=c];
      }
    }
    for (long x=*pb/(*pa);x;x>>=1)
      if (x&1){
        *pb-=*pa;
        *pa<<=1;
        fprintf(f,"%d %d\n",ib+1,ia+1);
        if (!*pb){
          a=ia; b=ic;
          return;
        }
      }
      else{
        *pc-=*pa;
        *pa<<=1;
        fprintf(f,"%d %d\n",ic+1,ia+1);
        if (!*pc){
          a=ia; b=ib;
          return;
        }
      }
  }
}

void ComputeAndWriteSolution(void){
  int a=0,b=1;
  f=fopen("GOLD.OUT","wt");
  for (int c=2;c<n;c++)
    ThreePiles(a,b,c);
  fclose(f);
}

void main(void){
  ReadData();
  ComputeAndWriteSolution();
}
```

Au rezolvat corect:

Cristian Alexandrescu, Botoşani
Csaba Andras, Oradea
Marius Andrei, Bucureşti
Mugurel Ionuţ Andreica, Bucureşti
Liviu-Cosmin Andreicuţ, Bucureşti
Victor Asavei, Dr. Tr. Severin
Mihai-Ioan Bădilă, Cugir
Cristian Băicoianu, Ploieşti
Ciprian-Nicolae Baicu, Dr. Tr. Severin
Valentin Bisa, Lugoj
Denis Bogdănaş, Iaşi
Ciprian Cană, Vaslui
Adrian Cârcu, Bistriţa
Alexandra-Elena Constantin, Ploieşti
Horea Coroiu, Cluj-Napoca
Daniel Crişan, Râmnicu Vâlcea
Andrei Csibi, Cluj-Napoca
Vlad Dascălu, Bacău
Andrei David, Râmnicu Vâlcea
Radu Dondera, Bucureşti
George Drumea, Bucureşti
Octavian-Daniel Dumitran, Bucureşti
Dan Ghinea, Bucureşti
Claudiu Gruia, Bucureşti
Mihai Hărănguş, Cluj-Napoca
Ioana-Maria Ileană, Alba Iulia
Liviu Lalescu, Craiova
Codruţ-Lucian Lazăr, Câmpeni
Bogdan Lucaciu, Dr. Tr. Severin
Maximilian Machedon, Bucureşti
Emilian Miron, Bacău
Cosmin-Silvestru Negruseri, Bistriţa
Bogdan Nicolae, Sibiu
Sorin Otescu, Braşov
Mihai-Vlad Pantiş, Cluj-Napoca
Csaba Patcas, Oradea
Mihai Pătraşcu, Craiova
Liviu Păunescu, Constanţa
Gabriel Pârvan, Râmnicu Sărat
Bogdan-Milovan Piloga, Timişoara
Valentin Pop, Carei
Mihail Popa, Bucureşti
Dan Popovici, Arad
Cosmin Răianu, Constanţa
Paul Rusu, Cluj-Napoca
Bogdan Stan, Câmpina
Mihai Stroe, Bucureşti
Radu Ştefan, Braşov
Cristian Toth, Lugoj
Silviu-Gabriel Udrea, Târgu Jiu
Radu Vătavu, Suceava
Vlad Vâlceanu, Arad
Victor Vernescu, Constanţa

Runda 3, problema 2

Fantome

Această problemă a fost rezolvată corect de către 43 dintre participanții la cea de-a treia rundă a *Concursului de Programare Agora*. O rezolvare pentru o problemă echivalentă a fost publicată în numărul 2/1994 al *Gazetei de Informatică*.

Enunțul problemei echivalente era următorul: *O placă dreptunghiulară de dimensiuni* $L1 \times L2$ ($L1 \le 600$, $L2 \le 300$) *este împărțită într-o rețea ortogonală de pătrate de latură 1, în care sunt marcate* N *puncte albe și* N *puncte negre, toate având coordonate întregi. Oricare trei dintre cele* $2 \cdot N$ *puncte nu sunt coliniare. În scopul obținerii unui circuit imprimat prin corodarea plăcii, fiecare punct alb urmează a fi unit printr-un segment de dreaptă cu un punct negru oarecare, dar astfel încât oricare două dintre cele* N *segmente să nu se intersecteze.*

Singurele diferențe, care au apărut în cazul problemei propuse la runda a treia, sunt reprezentate de stabilirea dimensiunilor tablei la 1.000×1.000 și posibilitatea ca punctele să aibă coordonate reale cu trei zecimale. Putem înmulți fiecare valoare cu 1.000 și obținem o placă de dimensiuni $1.000.000 \times 1.000.000$.

În *Gazeta de Informatică* 2/1994 erau prezentate două soluții pentru rezolvarea problemei. Prima se baza pe metoda *backtracking*: se încearcă o conectare a punctului alb (vânătorului) curent cu un punct negru (fantomă) neconectat încă, astfel încât segmentul (raza) format(ă) să nu se intersecteze cu nici unul dintre segmentele (razele) deja formate. Este evident că o astfel de soluție nu se va încadra în limita de timp admisă, așadar nu vom mai insista asupra ei.

Cea de-a doua soluție folosește un algoritm care evită metoda *backtracking*. Pentru început vom conecta punctele într-un anumit mod. De exemplu, am putea conecta punctele (vânătorii și fantomele) în ordinea dată la intrare: primul punct alb (vânător) cu primul punct negru (fantomă) etc.

Vom indentifica prin V_i punctele albe (vânătorii) și cu F_i punctele negre (fantomele). Vom verifica apoi dacă există două segmente V_1F_1 și V_2F_2 care se intersectează. În acest caz conexiunile se vor modifica și noile segmente vor fi V_1F_2 și V_2F_1. Este evident că aceste segmente nu se vor mai intersecta.

Vom demonstra în continuare că, prin schimbarea conexiunilor, suma lungimilor celor două segmente se reduce. Vom nota cu O intersecția segmentelor V_1F_1 și V_2F_2. Știm că într-un triunghi lungimea unei laturi este întotdeauna mai mică decât suma lungimilor celorlalte două. Așadar, în triunghiul V_1OF_2 avem $V_1F_2 < V_1O + OF_2$, iar în triunghiul V_2OF_1 avem $V_2F_1 < V_2O + OF_1$. Adunând cele două inegalități obținem $V_1F_2 + V_2F_1 < V_1O + OF_1 + V_2O + OF_2$, adică $V_1F_2 + V_2F_1 < V_1F_1 + V_2F_2$.

Așadar, după fiecare modificare a conexiunilor suma totală a lungimilor segmentelor se reduce. Deoarece această sumă nu va fi niciodată negativă, înseamnă că după un număr finit de pași nu vom mai avea intersecții.

Se conturează astfel următorul algoritm de rezolvare a problemei: *atâta timp cât există două segmente care se intersectează se modifică conexiunile pentru punctele corespunzătoare.*

Aceasta a fost soluția prezentată de majoritatea celor care au obținut punctaj maxim la această problemă. Acest lucru s-a datorat faptului că redacția a considerat că o rezolvare corectă care nu folosește metoda *backtracking* merită 100 de puncte. Totuși, puteau fi găsite destul de ușor date de test pentru care programele să nu se încadreze în limita de timp admisă. Ordinul de complexitate al algoritmului prezentat anterior este liniar în suma lungimilor segmentelor, dar nu este polinomial în numărul de puncte, deoarece după modificarea conexiunilor pentru două segmente, segmentele nou formate se pot intersecta cu alte segmente, deci numărul de intersecții nu se reduce neapărat, ci numai suma totală a lungimilor.

Enunțul problemei, așa cum a fost el publicat în numărul 10/8 al *GInfo* a fost preluat din cartea ***Introducere în algoritmi***. Problema apare la sfârșitul capitolului 35 (*Geometrie computațională*) - pagina 783. În *CLR* (aceasta este denumirea folosită pe plan internațional pentru cartea ***Introducere în algoritmi***; denumirea provine de la inițialele celor trei autori: *Cormen*, *Leiserson*, *Rivest*) apar următoarele indicații pentru rezolvarea problemei:

- *Demonstrați că există o linie care trece printr-un vânător de fantome și printr-o fantomă, astfel încât numărul vânătorilor de fantome aflați de o parte a liniei este egal cu numărul fantomelor aflate de cealaltă parte. Descrieți cum se poate determina o astfel de linie într-un timp* $O(n \cdot \log n)$.
- *Se cere un algoritm de timp* $O(n^2 \cdot \log n)$ *care să echivaleze numărul vânătorilor de fantome cu numărul fantomelor în așa fel încât să nu se intersecteze nici o rază.*

Să prespunem că am reușit să determinăm linia cerută la prima indicație. Am putea acorda un premiu celui care

va găsi un algoritm de rezolvare folosind astfel de linii. Noi nu am reușit.

Din păcate, varianta în limba română a *CLR* conține o gravă eroare de traducere care probabil că a creat confuzii pentru mulți dintre cei care au încercat să rezolve această problemă bazându-se pe aceste indicații. Traducerea corectă a primei indicații este următoarea: *demonstrați că există o linie care trece printr-un vânător de fantome și printr-o fantomă, astfel încât numărul vânătorilor de fantome aflați de o parte a liniei este egal cu numărul fantomelor aflate de aceeași parte.*

Acum este destul de simplu să găsim un algoritm, dacă am determinat o astfel de dreaptă. Pentru rezolvare vom folosi metoda *divide et impera*. Dreapta va împărți vânătorii și fantomele în două mulțimi fiecare dintre ele conținând un număr egal de vânători și fantome. Așadar, același algoritm poate fi aplicat pentru cele două mulțimi fără a exista pericolul ca vreun segment trasat în una dintre mulțimi să se intersecteze cu un segment trasat în cealaltă mulțime sau cu segmentul care unește cele două puncte care determină dreapta care separă mulțimile.

Problema mai dificilă este determinare dreptei care are proprietatea cerută. Ordinul de complexitate $O(n \cdot \log n)$ indicat arată că este foarte probabil să folosim o anumită sortare pentru a determina acea dreaptă. Va trebui să folosim un algoritm performant cum ar fi *heapsort* sau *mergesort* pentru a ne încadra în ordinul de complexitate cerut.

Vom considera punctul `P` având coordonata `y` cea mai mică, indiferent dacă reprezintă poziția unui vânător sau a unei fantome.

Vom sorta toate celelalte puncte în funcție de unghiul format de dreapta care unește `P` de acest punct și dreapta orizontală care trece prin `P`.

Vom avea doi indicatori `V` și `F`, primul va indica numărul vânătorilor luați în considerare, iar al doilea numărul fantomelor.

Dacă punctul `P` reprezintă o fantomă atunci inițializăm `V` cu `0` și `F` cu `1`, iar dacă reprezintă un vânător inițializăm `V` cu `1` și `F` cu `0`.

Vom considera pe rând punctele sortate; dacă punctul curent reprezintă un vânător incrementăm indicatorul `V`, iar dacă reprezintă o fantomă incrementăm indicatorul `F`.

În momentul în care vom avea `V = F` vom ști că sub dreapta care unește punctul `P` cu punctul curent se află un număr egal de vânători și fantome.

Vom demonstra în continuare că, la un moment dat, vom ajunge în situația în care `V = F`. Presupunem, fără a restrânge generalitatea, că în punctul `P` se află un vânător.

Dacă în primul punct se află o fantomă atunci ajungem în situația `V = F = 1`, deci am determinat deja o dreaptă cu proprietatea cerută.

Dacă în ultimul punct se află o fantomă atunci, fie am găsit o dreaptă anterior, fie acest ultim punct și punctul `P` determină dreapta pe care o căutăm.

Așadar, în cazul în care primul sau ultimul punct reprezintă o fantomă este rezolvat.

Să presupunem acum că atât primul, cât și ultimul punct reprezintă un vânător. În acest caz inițial vom avea `V = 1` și `F = 0` (`V - F = 1`), iar înaintea considerării ultimului punct `V = n - 1` și `F = n` (`V - F = -1`).

Deoarece valoarea `V - F` nu se poate modifica decât cu `1` la fiecare pas, pentru a ajunge de la `1` la `-1` ea trebuie neapărat să treacă și prin `0`. Așadar, la un moment dat ne vom afla în situația pentru care `V = F`, deci vom avea dreapta cerută.

Datorită faptului că după sortare nu se face decât o singură parcurgere (într-un timp $O(n)$) a punctelor, ordinul de complexitate rămâne $O(n \cdot \log n)$.

În total trebuie să determinăm n astfel de drepte, deci ordinul de complexitate al algoritmului devine $O(n^2 \cdot \log n)$.

La implementare am folosit numere de la `0` la `n - 1` pentru a identifica vânătorii și numere de la `n` la `2·n - 1` pentru a identifica fantomele.

Să presupunem că în sistemul de coordonate cu originea în `P`, avem două puncte `A` și `B` de coordonate (x_1, y_1) și (x_2, y_2). Pentru a sorta punctele în funcție de unghi avem trei cazuri:

- Dacă punctele `A` și `B` se află în primul cadran al sistemului de coordonate cu originea în `P`, atunci punctul `A` se află înaintea punctului `B` în șirul sortat dacă și numai dacă $y_1/x_1 < y_2/x_2$, adică $y_1 \cdot x_2 < y_2 \cdot x_1$, deoarece coordonatele orizontale sunt pozitive. (Am înlocuit împărțirile cu înmulțiri pentru a mări viteza de execuție și a evita împărțirea cu zero.)
- Dacă punctele `A` și `B` se află în cadrane diferite, atunci punctul din primul cadran îl va precede pe cel din al doilea cadran.
- Dacă punctele `A` și `B` se află în al doilea cadran al sistemului de coordonate cu originea în `P`, atunci punctul `A` se află înaintea punctului `B` în șirul sortat dacă și numai dacă $y_1/x_1 < y_2/x_2$, adică $y_1 \cdot x_2 > y_2 \cdot x_1$, deoarece coordonatele orizontale sunt negative.

Datorită faptului că am ales punctul `P` având coordonata `y` minimă, punctele `A` și `B` nu se pot afla în al treilea sau al patrulera cadran.

Vă prezentăm în continuare versiunea oficială pentru rezolvarea acestei probleme.

Listing GHOSTS.CPP

```
#include <stdio.h>

float x[1000],y[1000];
   // coordonatele vanatorilor si fantomelor
int n, // numarul de vanatori si fantome
    pair[500], // pair[i] va indica fantoma
               // ucisa de vanatorul i
    p[1000]; // sir de indici folosit pentru
             // sortare si prelucrare

void ReadData(void){
  FILE *f=fopen("GHOSTS.IN","rt");
```

```
  fscanf(f,"%d",&n);
  for (int i=0;i<n<<1;i++)
    fscanf(f,"%f%f",&x[i],&y[i]);
  fclose(f);
}

int FindMinimum(int left,int right){
// determina punctul din multumimea curenta
// care are coordonata y minima
  int ind=left;
  float min=y[p[left]];
  for (int i=left+1;i<=right;i++)
    if (min>y[p[i]])
      min=y[p[ind=i]];
  return ind;
}

int HigherAngle(int left,int p1,int p2){
// functia folosita pentru a stabili relatia
// de ordine dintre doua puncte
  if (x[p[p1]]-x[p[left]]>=0 &&
      x[p[p2]]-x[p[left]]>=0 &&
      (x[p[p1]]-x[p[left]])*
      (y[p[p2]]-y[p[left]])<
      (x[p[p2]]-x[p[left]])*
      (y[p[p1]]-y[p[left]]))
    return 1; // puncte in primul cadran
  if (x[p[p1]]-x[p[left]]>=0 &&
      x[p[p2]]-x[p[left]]>=0 &&
      (x[p[p1]]-x[p[left]])*
      (y[p[p2]]-y[p[left]])<
      (x[p[p2]]-x[p[left]])*
      (y[p[p1]]-y[p[left]]))
    return 1; // puncte in al doilea cadran
  if (x[p[p1]]-x[p[left]]<=0 &&
      x[p[p2]]-x[p[left]]>=0)
    return 1; // puncte din cadrane diferite
  return 0;
}

void Sift(int left,int right,int k){
// functia pentru scufundarea unui element
// intr-un heap
  int son;
  if ((k<<1)+left<=right){
    son=k<<1;
    if (((k<<1)+left)<right &&
        HigherAngle(left,left+1+(k<<1),
                    left+(k<<1)))
      son++;
    if (HigherAngle(left,k+left,son+left))
      son=0;
  }
  else
    son=0;
  while (son){
    int aux=p[k+left];
    p[k+left]=p[son+left];
    p[son+left]=aux;
    k=son;
    if ((k<<1)+left<=right){
      son=k<<1;
      if (((k<<1)+left)<right &&
        HigherAngle(left,left+1+(k<<1),
                    left+(k<<1)))
        son++;
      if (HigherAngle(left,k+left,son+left))
        son=0;
    }
    else
      son=0;
  }
}

void HeapSort(int left,int right){
// crearea unui heap pe baza unui vector
  for (int i=(right-left)>>1;i;i--)
    Sift(left,right,i);
// sortarea heap-ului
  for (i=right-left;i>=2;i--){
    int aux=p[left+1];
    p[left+1]=p[left+i];
    p[left+i]=aux;
    Sift(left,left+i-1,1);
  }
}

int FindLine(int left,int right){
// determina punctul cu care trebuie uni
// punctul de coordonata y minima pentru a
// determina o dreapta astfel incat numarul
// de vanatori care se afla de o parte a
// dreptei sa fie egal cu numarul de fantome
// care se afla de aceeasi parte a dreptei
  int v=0,f=0;
  for (int i=left;;i++){
    (p[i]<n)?v++:f++;
    if (v==f)
      return i;
  }
}

void DivideEtImpera(int left,int right){
// functia care implementeaza metoda divide
// et impera folosita pentru rezolvarea
// problemei
  if (left>=right)
    return; // multimea este vida
  // se alege punctul de coordonata y minima
  int ind=FindMinimum(left,right);
```

```
  // punctul determinat este mutat pe prima
  // pozitie, iar punctul care se afla pe
  // aceasta pozitie este mutat in locul
  // ramas liber
  float aux=p[ind];
  p[ind]=p[left];
  p[left]=aux;
  // se sorteaza celelalte puncte in functie
  // de unghiul format de dreapta care le
  // uneste de punctul ales si dreapta
  // orizontala acre trece prin punctul ales
  HeapSort(left,right);
  // se determina linia care va imparti
  // multimea de puncte in doua multimi
  ind=FindLine(left,right);
  // se testeaza daca punctul ales reprezinta
  // pozitia unui vanator sau a unei fantome
  // si se realizeaza corespondenta, adica
  // este aleasa fantoma care va fi ucisa de
  // vanator
  if (p[ind]<n)
    pair[p[ind]]=p[left];
  else pair[p[left]]=p[ind];
  // se va folosi acelasi algoritm pentru
  // cele doua multimi de puncte in care a
  // fost impartita multimea initiala;
  // pentru aceasta se va autoapela functia
  // care implementeaza metoda divide et
  // impera
  DivideEtImpera(left+1,ind-1);
  DivideEtImpera(ind+1,right);
}

void KillGhosts(void){
  // se initializeaza sirul de indici
  for (int i=0;i<n<<1;i++)
    p[i]=i;
  // se apeleaza functia care implementeaza
  // metoda divide et impera pentru multimea
  // initiala de puncte
  DivideEtImpera(0,(n<<1)-1);
}

void WriteSolution(void){
  FILE *f=fopen("GHOSTS.OUT","wt");
  for (int i=0;i<n;i++)
    fprintf(f,"%d\n",pair[i]-n+1);
  fclose(f);
}

void main(void){
  ReadData();
  KillGhosts();
  WriteSolution();
}
```

Au rezolvat corect:

Marius Andrei, Bucureşti
Mugurel Ionuţ Andreica, Bucureşti
Liviu-Cosmin Andreicuţ, Bucureşti
Cristian Baicoianu, Ploieşti
Ciprian Baicu, Drobeta Turnu Severin
Livia Bana, Drobeta Turnu Severin
Valentin Bişa, Lugoj
Denis Bogdănaş, Iaşi
Vencel Bors, Oradea
Ioana-Violeta Brutaru, Hunedoara
Ciprian Cană, Vaslui
Adrian Cârcu, Bistriţa
Horea Coroiu, Cluj-Napoca
Daniel Crişan, Râmnicu Vâlcea
Vlad Dascălu, Bacău
Andrei David, Râmnicu Vâlcea
Laurent Demonet, Franţa
Radu Dondera, Bucureşti
George Drumea, Bucureşti
Vincent Groenhuis, Olanda
Ioana-Maria Ileană, Alba Iulia
Liviu Lalescu, Craiova
Maximilian Machedon, Bucureşti
Paul Marinescu, Buzău
Cosmin-Silvestru Negruseri, Bistriţa
Mihai-Vlad Pantiş, Cluj-Napoca
Gabriel Pârvan, Râmnicu Sărat
Mihai Pătraşcu, Craiova
Bogdan-Milovan Piloga, Timişoara
Mohammad Ali Safari Ghahsareh, Iran
Bogdan Stan, Câmpina
Mihai Stroe, Bucureşti
Radu Ştefan, Braşov
Radu Vătavu, Suceava
Victor Vernescu, Constanţa

Runda 3, problema 3

Unitate

Această problemă a fost rezolvată corect de către 43 dintre participanții la cea de-a treia rundă a *Concursului de Programare Agora*. O rezolvare a acestei probleme poate fi găsită în cartea *Proiectarea și implementarea algoritmilor* scrisă de *Mihai Oltean*.

Autorul propune o soluție folosind programarea dinamică. Se construiește o matrice B de dimensiuni $m \times n$ în care elementul b_{ij} reprezintă numărul minim de unități care trebuie adăugate pentru ca toate elementele matricei inițiale care se află pe primele i linii și primele j coloane să aibă valoarea 1.

Cu alte cuvinte, elementul b_{ij} conține numărul de unități care trebuie adăugate pentru ca submatricea care are colțul stânga-sus în poziția (1, 1) și colțul dreapta-jos în poziția (i, j) să conțină numai elemente cu valoarea 1.

Modul de calcul al elementelor matricei B este următorul:

- Elementele de pe prima linie se calculează pe baza formulelor: $b_{11} = 1 - a_{11}$ și $b_{1,i} = 1 - a_{1,i} + c_{1,i-1}$ pentru $i > 1$.
- Elementele de pe prima coloană se calculează pe baza unor formule asemănătoare: $b_{11} = 1 - a_{11}$ și $b_{i,1} = 1 - a_{i,1} + c_{i-1,1}$ pentru $i > 1$.
- Dacă $a_{ij} = 1$, atunci nu mai trebuie adăugată nici o unitate pentru a obține cifra 1 pe poziția (i, j), deci vom avea
$$c_{i,j} = c_{i,j-1} + c_{i-1,j} - c_{i-1,j-1}.$$
- Dacă $a_{ij} = 0$, atunci trebuie adăugate un număr minim de unități pentru a obține cifra 1 pe poziția (i, j). Dacă notăm cu k minimul dintre i și j, vom adăuga valoarea 2^{k-1}, deci formula va deveni
$$c_{i,j} = c_{i,j-1} + c_{i-1,j} + c_{i-1,j-1} + 2^{k-1}.$$

Analizând această soluție observăm că nu se adaugă unități decât în cazul în care întâlnim un element $a_{ij} = 0$, iar numărul de unități adăugate în acest caz este $2^{\min(i,j)-1}$. Astfel algoritmul de programare dinamică poate fi înlocuit cu următorul:

```
algoritm unitate
  total ← 0
  pentru i ← 1, m execută
    pentru i ← 1, n execută
      dacă aij=0 atunci
          total ← total + 2^min(i,j)-1
      sfârșit dacă
    sfârșit pentru
  sfârșit pentru
  returnează total
sfârșit algoritm
```

Datorită faptului că matricea poate avea până la 100 de linii sau coloane, s-ar putea să trebuiască să adunăm valori foarte mari cum ar fi 2^{99}. Datorită acestui fapt va trebui să implementăm operațiile de adunare și înmulțire pentru numere întregi foarte mari.

Pentru a mări timpul de execuție nu vom efectua adunările în momentul în care vom întâlni un element cu valoarea 0, ci vom păstra un vector v care să indice de câte ori va trebui să adunăm valoarea 2^i. O putere a lui 2 poate fi foarte ușor determinată pe baza puterii anterioare și pentru fiecare putere vom înmulți rezultatul cu valoarea corespunzătoare din vectorul v, apoi îl vom aduna la valoarea care va indica în final numărul de unități care trebuie adăugate.

Soluția oficială a acestei probleme este prezentată în continuare:

Listing UNIT.CPP

```
#include <stdio.h>

int  m,n,v[100],
     units[50],dim,
     p2[50],dim2,
     x[50],dimx;

void ReadData(void){
  FILE *f=fopen("UNIT.IN","rt");
  fscanf(f,"%d%d",&m,&n);
  int x;
  for (int i=0;i<m;i++)
    for (int j=0;j<n;j++){
      fscanf(f,"%d",&x);
      if (!x)
        v[((i<j)?i:j)]++;
    }
}

void MultiplyBy2(void){
  int x,t=0;
  for (int i=0;i<dim2;i++){
    x=(p2[i]<<1)+t;
```

```
    p2[i]=x%10;
    t=x/10;
  }
  if (t)
    p2[dim2++]=1;
}

void Multiply(int v){
  int t=0;
  for (int i=0;i<dimx;i++)
    x[i]=0;
  for (i=0;i<dim2;i++){
    x[i]=(p2[i]*v+t)%10;
    t=(p2[i]*v+t)/10;
  }
  dimx=dim2;
  while (t){
    x[dimx++]=t%10;
    t/=10;
  }
}

void Add(void){
  int v,t=0;
  dim=((dim>dimx)?dim:dimx);
  for (int i=0;i<dim;i++){
    v=units[i]+x[i]+t;
    units[i]=v%10;
    t=v/10;
  }
  if (t)
    units[dim++]=1;
}

void BuildSolution(void){
  dim2=dimx=dim=p2[0]=1;
  for (int i=0;i<((m<n)?m:n);i++){
    Multiply(v[i]);
    Add();
    MultiplyBy2();
  }
}

void WriteSolution(void){
  FILE *f=fopen("UNIT.OUT","wt");
  for (int i=dim-1;i>=0;i--)
    fprintf(f,"%d",units[i]);
  fclose(f);
}

void main(void){
  ReadData();
  BuildSolution();
  WriteSolution();
}
```

Au rezolvat corect:

Csaba Andras, Oradea
Marius Andrei, Bucureşti
Mugurel Ionuţ Andreica, Bucureşti
Liviu-Cosmin Andreicuţ, Bucureşti
Victor Asavei, Drobeta Turnu Severin
Cristian Băicoianu Ploieşti
Ciprian Baicu, Drobeta Turnu Severin
Livia Bana, Drobeta Turnu Severin
Valentin Bisa, Lugoj
Denis Bogdănaş, Iaşi
Ciprian Cană, Vaslui
Adrian Cârcu, Bistriţa
Daniel Crişan, Râmnicu Vâlcea
Andrei David, Râmnicu Vâlcea
Laurent Demonet, Franţa
George Drumea, Bucureşti
Octavian-Daniel Dumitran, Bucureşti
Dan Ghinea, Bucureşti
Aurelian Ghiţă, Buzău
Vincent Groenhuis, Olanda
Claudiu Gruia, Bucureşti
Ioana-Maria Ileană, Alba Iulia
Liviu Lalescu, Craiova
Codruţ-Lucian Lazăr, Câmpeni
Bogdan Lucaciu, Drobeta Turnu Severin
Maximilian Machedon, Bucureşti
Andrei Markovits, Satu Mare
Cosmin-Silvestru Negruseri, Bistriţa
Bogdan Nicolae, Sibiu
Sorin Otescu, Braşov
Mihai-Vlad Pantiş, Cluj-Napoca
Flaviu Paşca, Bistriţa
Mihai Pătraşcu, Craiova
Liviu Păunescu, Constanţa
Dan Popovici, Arad
Mohammad Ali Safari Ghahsareh, Iran
Lucian-Raul Silistru, Vaslui
Bogdan Stan, Câmpina
Mihai Stroe, Bucureşti
Radu Ştefan, Braşov
Cristian Toth, Lugoj
Vlad Vâlceanu, Arad
Radu Vătavu, Suceava

Runda 5

Vă prezentăm în continuare cele trei probleme de la a cincea rundă a Concursului de Programare Agora. Concurenții care nu s-au înscris până la data de 20 Februarie 2001 pot participa la concurs începând cu această rundă, dacă trimit mesajul de înscriere până pe data de 15 Martie 2001.

CPAR5P1: Alb și negru

Nume fișier sursă: `BW.PAS`, `BW.C` sau `BW.CPP`
Nume fișier de intrare: `BW.IN`
Nume fișier de ieșire: `BW.OUT`

Descriere: Pe o tablă liniară cu $2 \cdot n + 1$ căsuțe se află n bile albe (pe primele n poziții) și n bile negre (pe ultimele n poziții), căsuța din mijloc fiind liberă. O bilă albă poate fi deplasată în căsuța din dreapta ei dacă aceasta este liberă (operația `W`) sau poate sări peste bila din dreapta ei, dacă a doua căsuță din dreapta este liberă (operația `w`). O bilă neagră poate fi deplasată în căsuța din stânga ei dacă aceasta este liberă (operația `B`) sau poate sări peste bila din stânga ei, dacă a doua căsuță din stânga este liberă (operația `b`). După un număr oarecare de astfel de operații bilele albe trebuie să ajungă pe ultimele n poziții, iar cele negre pe primele n poziții.

Date de intrare: Fișierul de intrare va conține pe prima linie valoarea n ($1 \le n \le 1000$).

Date de ieșire: Fișierul de ieșire va conține, pe un singur rând și fără spații, codificarea operațiilor efectuate.

Exemplu:

`BW.IN`	`BW.OUT`
2	WbBwwBbW

Timp de execuție: 1 secundă/test.

CPAR5P2: Coeficient

Nume fișier sursă: `COEF.PAS`, `COEF.C` sau `COEF.CPP`
Nume fișier de intrare: `COEF.IN`
Nume fișier de ieșire: `COEF.OUT`

Descriere: Se dau n ($2 \le n \le 1000$) orașe care sunt legate prin m ($2 \le m \le 5000$) străzi cu dublu sens. Fiecare stradă are atașat un coeficient întreg c ($1 \le c \le 1000$). Între două orașe poate există cel mult o stradă. Coeficientul unui traseu este dat de produsul coeficienților străzilor care îl formează. Se cere să se determine coeficientul minim al unui traseu de la orașul 1 la orașul n. Se știe că există întotdeauna un astfel de traseu.

Date de intrare: Prima linie a fișierului de intrare conține numărul n al orașelor și numărul m al străzilor separate printr-un singur spațiu. Următoarele m linii conțin câte trei numere x, y și z (separate prin spații) cu semnificația: între orașele x și y există o stradă de coeficient z.

Date de ieșire: Fișierul de ieșire va conține coeficientul minim al unui traseu între orașele 1 și n.

Exemplu:

`COEF.IN`	`COEF.OUT`
5 6	16
1 2 2	
1 4 2	
2 3 5	
3 4 2	
3 5 4	
4 5 9	

Timp de execuție: 1 secundă/test.

CPAR5P3: Matrice

Nume fișier sursă: `MATRIX.PAS`, `MATRIX.C` sau `MATRIX.CPP`
Nume fișier de intrare: `MATRIX.IN`
Nume fișier de ieșire: `MATRIX.OUT`

Descriere: Se consideră o matrice având dimensiunile $m \times n$ ($1 \le m, n \le 500$) care are toate elementele egale cu 1. O operație constă în schimbarea semnului tuturor elementelor de pe o linie sau de pe o coloană. În urma mai multor astfel de operații trebuie obținute exact k ($1 \le k \le 250.000$) valori -1 în întreaga matrice.

Date de intrare: Prima linie a fișierului de intrare conține valorile m și n, separate printr-un spațiu, reprezentând dimensiunile matricei. A doua linie conține numărul k de valori -1 care trebuie obținute.

Date de ieșire: În cazul în care nu există soluție, fișierul de ieșire va conține doar valoarea 0. Dacă există soluție, atunci fișierul va conține câte o linie pentru fiecare operație efectuată. O operație este descrisă printr-un caracter c, urmat de un număr x (fără ca acestea să fie separate prin spații). Caracterul c va avea valoarea `L` dacă operația este efectuată asupra unei linii și valoarea `C` dacă este efectuată asupra unei coloane. Valoarea x va indica numărul liniei sau coloanei asupra căreia este efectuată operația (numerotarea începe de la 1).

Exemplu:

`MATRIX.IN`	`MATRIX.OUT`
3 3	L2
4	C2

Timp de execuție: 1 secundă/test.

PUNCTAJE modificate

În urma rezolvării contestațiilor pentru problemele de la primele două runde, punctajele unor concurenți au fost modificate. De asemenea, am hotărât să acordăm puncte și acelor concurenți care au trimis fișiere .BAT care nu aveau o structură corectă sau conțineau anumite erori.

Marius Andrei,
București
RUNDA 1
Punctaj inițial: 0
Punctaj modificat: 225

Liviu Andreicuț,
București
RUNDA 1
Punctaj inițial: 39
Punctaj modificat: 42

Cristian Bojinovici,
București
RUNDA 2
Punctaj inițial: 0
Punctaj modificat: 137

Bogdan Chirilă,
Botoșani
RUNDA 1
Punctaj inițial: 0
Punctaj modificat: 97

Alexandra Constantin,
Ploiești
RUNDA 2
Punctaj inițial: 87
Punctaj modificat: 98

Alina Fești,
Cluj-Napoca
RUNDA 2
Punctaj inițial: 0
Punctaj modificat: 13

Andrei Huțu,
Bacău
RUNDA 1
Punctaj inițial: 19
Punctaj modificat: 47

Maximilian Machedon,
București
RUNDA 1
Punctaj inițial: 0
Punctaj modificat: 300

Mojca Miklavec,
Slovenia
RUNDA 1
Punctaj inițial: 187
Punctaj modificat: 207

Alexandru Mosoi,
Bacău
RUNDA 1
Punctaj inițial: 0
Punctaj modificat: 300

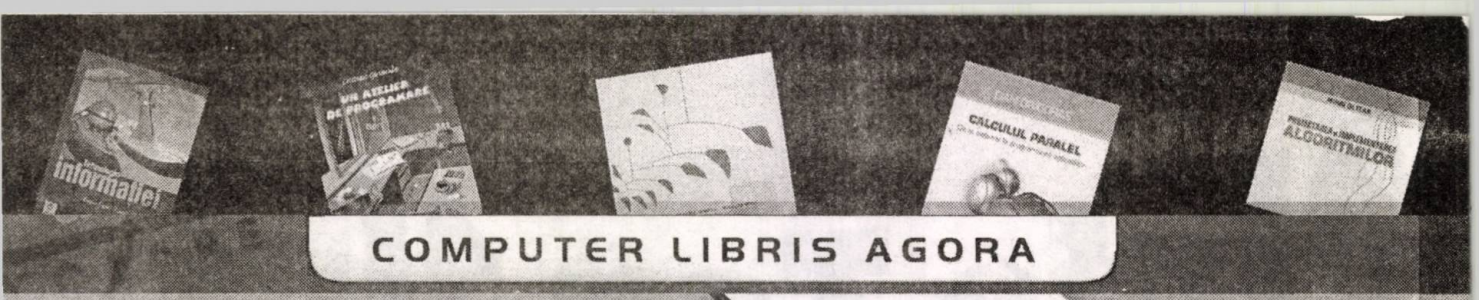
Bogdan Nicolae,
Sibiu
RUNDA 2
Punctaj inițial: 103
Punctaj modificat: 189

Bogdan Țira,
Cluj-Napoca
RUNDA 2
Punctaj inițial: 0
Punctaj modificat: 113

Ginfo Top 100

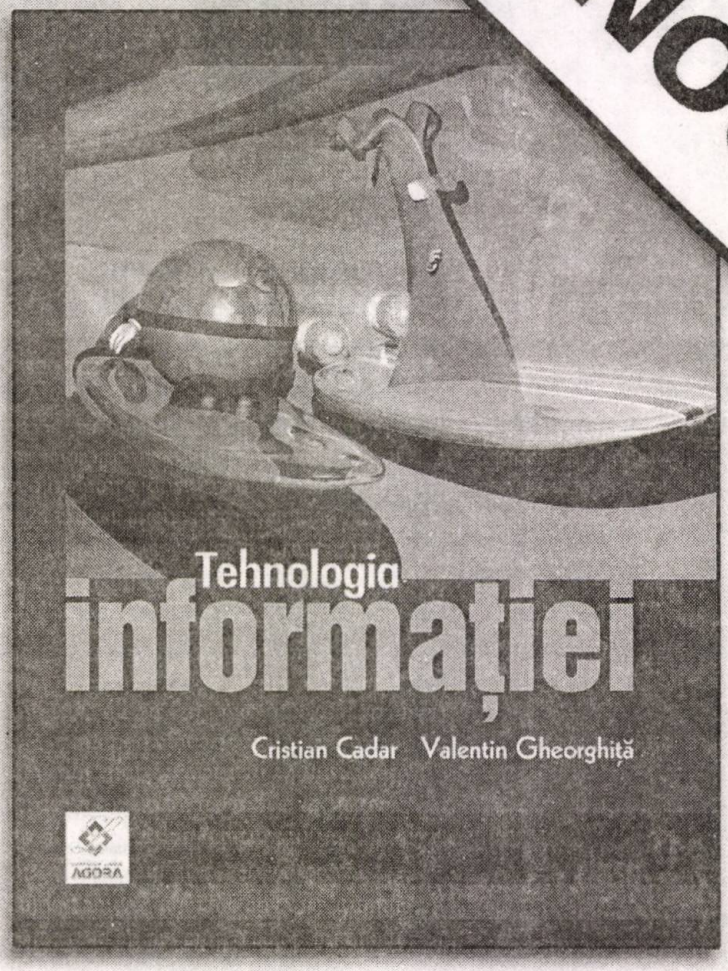
1. Ciprian Cană, Vaslui	886	6. Liviu Lalescu, Craiova	833
2. Maximilian Machedon, București	874	7. Cristian Băicoianu, Ploiești	830
3. George Drumea, București	871	8. Adrian Cârcu, Bistrița	824
4. Mihai Stroe, București	863	8. Ioana-Maria Ileană, Alba Iulia	824
5. Radu Vătavu, Suceava	838	8. Radu Ștefan, Brașov	824

11. Gabriel Pârvan, Râmnicu Sărat	811	41. Bogdan Dumitru, Ploiești	590	71. Bogdan Lucaciu, Dr. Turnu Severin	312
12. Daniel Crișan, Râmnicu Vâlcea	809	42. Alexandra-Elena Constantin, Ploiești	571	72. Artur Kuczapski, Oradea	311
13. Denis Bogdănaș, Iași	806	43. Horia Ciurdar, Timișoara	563	73. Mihail Pit-Rada, Dr. Turnu Severin	310
14. Mihai Pătrașcu, Craiova	805	44. Cristian Toth, Lugoj	555	74. Laurent Demonet, Franța	306
15. M. A. Safari Ghahsareh, Iran	800	45. Andrei Csibi, Cluj-Napoca	538	75. Bogdan-Nicolae Șanta, Cluj-Napoca	290
16. Csaba Andras, Oradea	799	46. Csaba Patcas, Oradea	525	76. Alina Fesnic, Cluj-Napoca	284
17. Bogdan Stan, Câmpina	772	47. Victor Vernescu, Constanța	522	77. Lucian-Raul Silistru, Vaslui	282
18. Dan Ghinea, București	766	48. Codruț-Lucian Lazăr, Câmpeni	521	78. Cristian Bojinovici, București	266
19. Mugurel Ionuț Andreica, București	732	49. Răzvan Mușăloiu-Elefteri, București	516	78. Preslav Nakov, Bulgaria	266
19. Claudiu Gruia, București	732	50. Aurelian Ghiță, Buzău	488	80. Narayanan Arvind, India	264
21. Victor Asavei, Drobeta Turnu Severin	721	51. Paul Marinescu, Buzău	475	81. Livia Bana, Drobeta Turnu Severin	259
22. Vlad Vâlceanu, Arad	719	52. Bogdan-Milovan Piloga, Timișoara	470	82. Andrei Benea, Focșani	254
23. Cristian Alexandrescu, Botoșani	714	53. Alexandru Mosoi, Bacău	463	83. Cristian Ioniță, București	243
24. Ciprian Baicu, Drobeta Turnu Severin	706	54. Horea Coroiu, Cluj-Napoca	459	84. Petko Minkov, Bulgaria	241
25. Bogdan Nicolae, Sibiu	692	55. Cătălin Kesa, Bușteni	458	85. Silviu-Gabriel Udrea, Târgu Jiu	235
26. Octavian-Daniel Dumitran, București	688	56. Radu Dondera, București	446	86. Andrei Adler, București	228
27. Vincent Groenhuis, Olanda	687	57. Liviu-Cosmin Andreicuț, București	442	87. Radu Cocieru, Sibiu	221
28. Mihail Popa, București	672	58. Mihai Hărănguș, Cluj-Napoca	421	88. Aurel-Mihai Fulger, Constanța	216
29. Sorin Otescu, Brașov	664	58. V. Muthuramakrishnan, India	421	88. Edison Nica, Iași	216
30. Flaviu Pașca, Bistrița	663	60. Andrei Markovits, Satu Mare	419	90. Emilian Miron, Bacău	215
31. Marius Andrei, București	658	61. Rusu Paul, Cluj-Napoca	396	91. Răzvan Dinu, Ploiești	213
32. Dan Popovici, Arad	634	62. Andrei Giurgiu, București	367	92. Mojca Miklavec, Slovenia	207
33. Andrei David, Râmnicu Vâlcea	633	63. Arcadie Crăcan, Iași	355	93. Constantin Asofiei, Târgu Neamț	203
34. Vlad Dascălu, Bacău	632	64. Valentin Pop, Carei	351	93. Dao The Phuong, Vietnam	203
35. Liviu Păunescu, Constanța	619	65. Bogdan-Vlad Țira, Cluj-Napoca	349	95. Damir Korenčic, Croația	200
36. Mihai-Vlad Pantiș, Cluj-Napoca	606	66. Mihai Gojol, Cluj-Napoca	335	96. Daniel Comșa, Crăciunelu de Jos	193
37. Vencel Bors, Oradea	600	67. Adrian Cearnău, București	331	97. Petru-Simon Moț, Brăila	187
37. Cosmin-Silvestru Negruseri, Bistrița	600	68. Raluca Sauciuc, București	330	97. Andrei Tătăranu, Buzău	187
39. Valentin Bisa, Lugoj	599	69. Lucian Burja, Blaj	320	99. Ioana-Violeta Brutaru, Hunedoara	179
40. Cosmin Răianu, Constanța	596	70. Raluca Mușăloiu-Elefteri, București	316	100. Andrei-Liviu Dumbrava, Iași	177



COMPUTER LIBRIS AGORA

NOUTATE



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

HEAP-uri

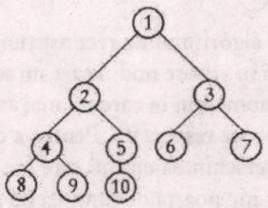
Mihai Scorțaru

În cadrul acestui articol vom prezenta una dintre cele mai interesante structuri de date care poate fi folosită pentru algoritmii de sortare și căutare. Aceasta se implementează relativ ușor și permite crearea de programe care se execută într-un timp mai scurt decât cele care folosesc alte metode pentru organizarea datelor.

Definiția heap-ului

Vom începe prin a defini structura de ***heap***. Denumirea este preluată din limba engleză și, în general, nu este folosită traducerea în limba română. O traducere aproximativă ar putea fi ***grămadă***; se mai folosește și termenul de ***ansamblu***.

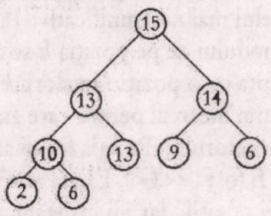
Un *heap* este un vector care poate fi considerat ca fiind un arbore binar după cum se poate vedea și în figura următoare:



Numărul care apare în fiecare nod indică poziția nodului respectiv în vector.

Așadar, rădăcina corespunde primului element al vectorului, fiul său stâng celui de-al doilea element, fiul drept celui de-al treilea și așa mai departe.

Considerăm vectorul `V={15,13,14,10,13,9,6,2,6}`. Arborele echivalent este următorul:



Pentru a obține vectorul corespunzător unui astfel de arbore, vom parcurge nivelurile arborelui începând cu rădăcina și fiecare nivel începând cu nodul cel mai din stânga.

Pentru ca un arbore binar să poată fi considerat un *heap*, toate nivelurile acestuia trebuie să fie complete, cu excepția ultimului care se completează începând cu nodul cel mai din stânga.

Înălțimea unui *heap* este înălțimea arborelui binar corespunzător. Înălțimea unui arbore este definită ca fiind egală cu adâncimea maximă a unui nod din arbore. Adâncimea unui nod este distanța dintre acel nod și rădăcina arborelui.

Un arbore complet de înălțime h are $1 + 2 + 4 + 8 + \ldots + 2^{h-1} = 2^h - 1$ noduri. Datorită faptului că ultimul nivel al unui *heap* poate să nu fie complet, numărul nodurilor unui *heap* este cuprins între 2^h și $2^{h+1}-1$. Reciproc, obținem că un *heap* cu n noduri are înălțimea $[\log_2 n]$.

Studiind organizarea arborelui, observăm că tatăl nodului corespunzător unei poziții $k > 1$ în vector este nodul corespunzător poziției $[k/2]$. Evident, tatăl rădăcinii nu există.

Reciproc, observăm că fiii nodului corespunzător poziției k sunt nodurile corespunzătoare pozițiilor $2 \cdot k$ (fiul stâng) și $2 \cdot k+1$ (fiul drept). În cazul în care valoarea $2 \cdot k$ este egală cu numărul de noduri, atunci fiul drept nu mai există, iar fiul stâng este ultimul nod al *heap*-ului. Dacă această valoare depășește numărul de noduri, atunci nici unul dintre fii nu mai există, nodul corespunzător poziției k fiind o frunză.

Cea mai importantă proprietate a *heap*-ului este aceea că valoarea oricărui nod nu poate fi mai mare decât valoarea nodului tată. Această proprietate face această structură foarte utilă pentru operațiile de căutare. Generalizând, obținem că valoarea corespunzătoare rădăcinii unui subarbore este mai mare sau egală cu oricare dintre valorile corespunzătoare oricărui nod din subarbore. De aici rezultă că rădăcina trebuie să conțină valoarea maximă a elementelor din vector.

Despre relațiile dintre valorile a două noduri, care nu sunt unul descendentul celuilalt, nu se poate face nici o afirmație. Cu alte cuvinte, faptul că un nod se află pe un nivel mai apropiat de rădăcină, nu înseamnă neapărat că el are o valoare mai mare decât nodurile aflate pe niveluri mai depărtate.

Această structură permite efectuarea foarte rapidă a anumitor operații:

- determinarea maximului valorilor dintr-un *heap*: $O(1)$;
- crearea unui *heap* dintr-un vector oarecare: $O(n)$;
- eliminarea unui element: $O(\log n)$;
- inserarea unui element: $O(\log n)$;
- sortarea unui vector: $O(n \cdot \log n)$;
- căutarea unei valori: $O(n)$.

Implementarea

Datorită echivalenței dintre un *heap* și un vector, datele nu vor fi reprezentate în memorie în formă arborescentă, ci în cea vectorială. Pentru implementare vom folosi limbajul C++. Vom considera *heap*-ul ca fiind un vector; tipul corespunzător va putea fi definit astfel:

```
typedef int Heap[MAX],
```

unde `MAX` reprezintă dimensiunea maximă a *heap*-ului. Deoarece dimensiunea *heap*-ului poate fi destul de mare, este recomandabil ca acesta să fie păstrat într-o variabilă globală și să nu fie transmis ca parametru pentru anumite funcții. Vom declara global un vector prin `Heap h;`.

Operațiile cu *heap*-uri trebuie efectuate astfel încât structura de *heap* să fie menținută în permanență, adică arborele să aibă toate nivelurile complete cu excepția ultimului, iar valoarea nici unui nod să nu depășească valoarea nodului părinte. Vom prezenta în continuare fiecare operație amintită.

Determinarea maximului

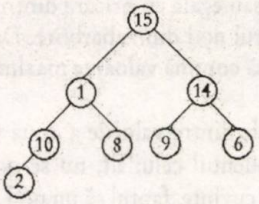
Datorită faptului că rădăcina corespunde primului element din vector și ea conține valoarea maximă, o funcție care determină valoarea maximă a elementelor dintr-un *heap*, va returna valoarea primului element al vectorului. Implementarea este următoarea:

```
int Max(){
  return h[1].
}
```

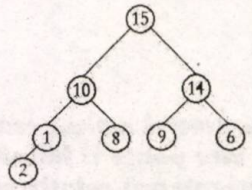
Din motive care vor fi explicate ulterior, poziția `0` nu este folosită.

Crearea unui heap dintr-un vector oarecare

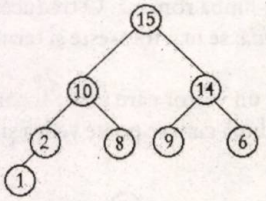
Pentu a prezenta această operație avem nevoie de o alta și anume reconstituirea structurii de *heap* atunci când avem un singur nod care nu respectă proprietatea de *heap* deoarece are o valoare mai mică decât unul dintre fiii săi. O asemenea situație este următoarea:



Nodul cu valoarea `1` nu se află pe poziția corectă deoarece valoarea sa este mai mică decât cel puțin una dintre valorile fiilor săi. În acest caz nodul va trebui coborât în arbore, fiind înlocuit cu unul dintre fiii săi. Dacă alegem nodul cu valoarea `8`, atunci structura de *heap* nu va fi corectă deoarece, după interschimbare, acest nod va avea ca fii un nod cu valoarea `10` și unul cu valoarea `1`, proprietatea de *heap* nefiind respectată. Dacă alegem nodul cu valoarea `10`, acesta va avea ca fii un nod cu valoarea `1` și unul cu valoarea `8`, proprietatea de *heap* fiind respectată în acest caz. Ca urmare, vom interschimba nodul având valoarea `1` cu nodul având valoarea `10`. Ajungem în următoarea situație:



Totuși, propritatea de *heap* nu este încă respectată, deoarece nodul cu valoarea `1` are un fiu cu valoarea `2`, deci cu valoare mai mare. De această dată avem un singur fiu, deci singura opțiune este de a interschimba cele două noduri. Ajungem în următoarea situație în care avem o structură de *heap* corectă:



În concluzie, algoritmul de reconstituire a proprietății de *heap* va coborî în arbore nodul care nu se află pe poziția corectă până în momentul în care nu mai avem fii sau proprietatea de *heap* este respectată. Pentru a coborî nodul în arbore îl vom interschimba cu fiul care are valoarea maximă. Această operație poartă denumirea de ***scufundare***.

Mai trebuie precizat faptul că, datorită relațiilor dintre pozițiile unui nod și cele ale fiilor săi, acestea pot fi determinate folosind doar operații pe biți (care sunt foarte rapide):

- Poziția fiului stâng al nodului de pe poziția k se obține prin deplasarea la stânga cu o poziție a valorii k: `k<<1`.
- Poziția fiului drept al nodului de pe poziția k se obține prin deplasarea la stânga cu o poziție a valorii k și setarea pe `1` a bitului celui mai nesemnificativ: `(k<<1)|1`.
- Poziția tatălui nodului de pe poziția k se obține prin deplasarea la dreapta cu o poziție a valorii k: `k>>1`.

Se observă acum motivul pentru care am ales să nu folosim poziția `0` a vectorului. Poziția fiului stâng al nodului de pe poziția `0` ar fi fost `0<<1=0`. Există artificii care permit și folosirea acestei poziții, dar nu insistăm asupra lor.

Vom implementa o funcție care realizează operația de scufundare. Aceasta va avea ca parametri dimensiunea *heap*-ului și poziția nodului care nu respectă proprietatea de *heap*.

```
void Scufunda(int n,int k){
  int aux,fiu=0;
```

```
  // se alege fiul cu valoarea mai mare
  if ((k<<1)<=n){
    fiu=k<<1;
    if ((k<<1)|1<n && h[(k<<1)|1]>H[k<<1])
      fiu|=1;
    // interschimbare numai daca este nevoie
    if (h[fiu]<=h[k]) fiu=0;
  }
  else fiu=0;
  while (fiu){
    // interschimbare
    aux=h[k];
    h[k]=h[fiu];
    h[fiu=k]=aux;
    // se alege urmatorul fiu
    if (k<<1<=n){
      fiu=k<<1;
      if ((k<<1)|1<n && h[(k<<1)|1]>H[k<<1])
        fiu|=1;
      // interschimbare numai daca este nevoie
      if (h[fiu]<=h[k]) fiu=0;
    }
    else fiu=0;
  }
}
```

Am preferat o variantă nerecursivă datorită vitezei mai mari de execuție. Acest algoritm funcționează numai dacă cei doi subarbori, care au ca rădăcină fiii nodului de pe poziția *k*, au o structură de *heap* corectă.

Vom prezenta acum modalitatea prin care această funcție de scufundare a unui element se folosește pentru a transforma un vector oarecare într-un *heap*.

Este evident faptul că frunzele arborelui au o structură de *heap* corectă. Ca urmare, vom putea apela funcția `Scufunda()` pentru nodurile imediat superioare frunzelor. După aceste apeluri, vom avea structuri de *heap* corecte pentru nodurile imediat superioare frunzelor. Vom trece apoi, pe rând, pe niveluri superioare până când vom ajunge la rădăcină.

În concluzie, va trebui să apelăm funcția `Scufunda()` începând cu nodul de pe poziția $[n/2]$ și continuând cu nodurile de pe pozițiile anterioare până când vom ajunge la prima poziție. Vom scrie o funcție care are ca parametru dimensiunea *h* a *heap*-ului.

```
void ConstruiesteHeap(int n){
  for (int i=(n>>1);i;Scufunda(n,i--));
}
```

În cele ce urmează vom determina complexitatea acestui algoritm de construire a unui *heap* dintr-un vector oarecare. La prima vedere, avem *n* noduri, care pot fi scufundate cel mult $[\log_2 n]$ niveluri deci, s-ar părea că avem un ordin de complexitate $O(n \cdot \log n)$. În realitate această determinare a complexității este eronată, deoarece nu ține cont de modul în care se fac scufundările. Frunzele reprezintă aproximativ jumătate din noduri și nu sunt scufundate deloc. Aproximativ un sfert dintre noduri se află imediat deasupra frunzelor și acestea sunt scufundate cel mult un nivel. O optime dintre noduri se află cu două niveluri deasupra frunzelor, deci sunt scufundate cel mult două niveluri. Continuând până la rădăcină, obținem un număr cel mult $\sum_{k=0}^{[\log_2 n]} \left(k \cdot \frac{n}{2^{k+1}} \right)$ scufundări. Dacă înălțimea arborelui este *h*, atunci *n* este, aproximativ, 2^h. Înlocuind în formulă obținem $\sum_{k=0}^{h} \left(k \cdot \frac{2^h}{2^{k+1}} \right)$; după efectuarea calculelor se obține ordinul de complexitate $O(2^h)$ pentru algoritmul de construire a unui *heap*.

Deoarece avem $h = [\log_2 n]$ vom obține ordinul de complexitate $O(2^{\log_2 n}) = O(n)$.

Eliminarea unui element

Pentru a efectua această operație avem nevoie, de asemenea, de o alta și anume reconstituirea structurii de *heap* atunci când avem un singur nod care nu respectă proprietatea de *heap*, deoarece are valoarea mai mare decât tatăl său. Este exact fenomenul invers celui prezentat în cazul operației de scufundare.

În acest caz nodul va trebui urcat în arbore. Vom presupune că restul *heap*-ului este corect.

Vom interschimba acest nod cu tatăl său. După interschimbare, nodul ajunge în poziția tatălui și subarborele având rădăcina în acest nod este un *heap* corect deoarece valoarea tatălui era mai mare decât cea a celuilalt fiu. Totuși, s-ar putea ca, după interschimbare, nodul să aibă un tată cu o valoare mai mică. În acest caz nodul va trebui din nou urcat în arbore. Procedeul va continua până când nodul curent devine rădăcina arborelui sau nu mai are un tată cu o valoare mai mică. Această operație poartă numele de *ridicare*.

Vom implementa o funcție care realizează operația de ridicare. Aceasta va avea ca parametri dimensiunea *heap*-ului *h* și poziția nodului care nu respectă proprietatea de *heap*.

```
void Ridica(int n,int k){
  int val=h[k];
  while (k>1 && val>h[k>>1])
    h[k]=h[k>>=1];
  h[k]=val;
}
```

Vom prezenta acum modul în care se folosește această funcție pentru a elimina un element dintr-un *heap*. După eliminarea unui element, în locul său apare un gol în care trebuie să amplasăm un alt element. Deoarece toate nivelurile (cu excepția ultimului) trebuie să fie complete, iar ultimul trebuie să fie ocupat în partea stângă, trebuie să eliminăm ultimul element din *heap*. Vom amplasa valoarea

corespunzătoare acestui element în locul gol și vom verifica dacă structura de *heap* este păstrată. Știm că singurul loc în care structura de *heap* s-ar putea să nu fie respectată este poziția elementului șters care a fost înlocuit cu ultimul element. Dacă valoarea este mai mare decât cea a tatălui, acest nod va trebui ridicat în arbore. Dacă nodul are o valoare mai mică decât cea a unui fiu, atunci nodul trebuie scufundat în arbore. În continuare vom prezenta o funcție care realizează eliminarea unui element dintr-un *heap*.

```
void Elimina(int& n,int k){
  h[k]=h[n--];
  if (k>1 && h[k]>h[k>>1]) Ridica(n,k);
  else Scufunda(n,k);
}
```

Deoarece o scufundare sau o ridicare se poate face cu cel mult h niveluri (h este înălțimea arborelui), iar h este $[\log_2 n]$, ordinul de complexitate al algoritmului de eliminare este $O(\log n)$.

După eliminare, dimensiunea *heap*-ului se reduce cu 1, deci variabila n trebuie decrementată.

Inserarea unui element

Dacă dorim să inserăm un nou element într-un *heap*, operația este mult mai simplă. Trebuie doar să îl amplasăm pe poziția $n+1$ a vectorului și să îl ridicăm până când ajunge în poziția corectă. O funcție care realizează operația de inserare este prezentată în continuare.

```
void Insereaza(int& n,int val){
  h[++n]=val;
  Ridica(n,n);
}
```

Elementul poate fi ridicat cel mult h niveluri, deci ordinul de complexitate al algoritmului este $O(\log n)$. După eliminare dimensiunea *heap*-ului crește cu 1, deci variabila n trebuie incrementată.

Sortarea unui vector

Algoritmul de sortare care folosește structura de *heap* poartă denumirea de ***heapsort***. Vom începe prin a construi un *heap* pentru vectorul pe care dorim să îl sortăm. După aceea eliminăm maximul și apoi refacem *heap*-ul. După refacere, al doilea element se află pe prima poziție a vectorului; îl eliminăm și refacem din nou *heap*-ul. Continuăm până când nu mai avem elemente în *heap*. Algoritmul de extragere a maximului are complexitatea $O(1)$, iar refacerea *heap*-ului necesită, așa cum vom vedea, o singură scufundare, deci are complexitatea $O(\log n)$. Având în vedere faptul că efectuăm n extrageri, complexitatea algoritmului *heapsort* este $O(n \cdot \log n)$.

Algoritmul poate fi implementat astfel încât să nu necesite spațiu suplimentar de memorie. După o eliminare a maximului, dimensiunea *heap*-ului se va reduce cu 1, deci ultima poziție va rămâne liberă. Deoarece în vectorul sortat maximul trebuie amplasat exact pe această poziție, la fiecare pas nu trebuie decât să interschimbăm prima și ultima poziție a *heap*-ului și apoi să scufundăm elementul care a devenit rădăcina arborelui.

Algoritmului *heapsort* poate fi implementat prin funcția HeapSort() prezentată în continuare:

```
void HeapSort(int n){
  ConstruiesteHeap(n);
  for (int i=n;i>=2;){
    int aux=h[1];
    h[1]=h[i];
    h[i]=aux;
    Scufunda(--i,1);
  }
}
```

Căutarea unei valori

Pentru această operație folosirea *heap*-urilor nu este cea mai indicată soluție, deoarece complexitatea nu poate fi redusă sub $O(n)$. Aceasta se datorează faptului că, deși putem fi siguri că un nod cu valoarea mai mică este descendent al unui nod cu valoarea mai mare, nu știm în care dintre cei doi subarbori să îl căutăm. Totuși, față de căutarea secvențială se poate aduce o îmbunătățire; dacă rădăcina unui subarbore are o valoare mai mică decât valoarea căutată, atunci vom ști că descendenții rădăcinii vor avea valori mai mici, deci nu mai are rost să căutăm în acel subarbore. Astfel, s-ar putea ca porțiuni mari din *heap* să nu mai trebuiască explorate, dar în cazul cel mai defavorabil, este necesară parcurgerea întregului *heap*. Algoritmul poate fi implementat recursiv, destul de simplu, după cum se poate vedea în continuare.

```
int Cauta(int nod,int val){
  if (nod>n || val>h[nod])
    return 0;
  if (val==h[nod])
    return 1;
  return Cauta(nod>>1,val) ||
    Cauta((nod>>1)|1,val);
}
```

Această funcție trebuie apelată cu valoarea 1 pentru parametrul nod.

Bibliografie

1. Thomas H. Cormen, Charles E. Leisserson, Ronald R. Rivest, *Introducere în algoritmi*, Editura *Computer Libris Agora*, Cluj - 2000
2. Cătălin Frâncu, *Psihologia concursurilor de informatică*, Editura *L&S*, București - 1997

Mihai Scorțaru este redactor-șef adjunct al GInfo. Poate fi contactat prin e-mail la adresa **skortzy@xnet.ro**.

Secvența de SUMĂ MAXIMĂ

Clara Ionescu

Frumusețea problemelor de programare se ascunde în rezolvarea lor. O problemă care se rezolvă în multe linii sursă, cu un algoritm banal, nu prea are șanse să se remarce în rândul rezolvitorilor de probleme de algoritmică. Răsfoind notițe făcute "de demult" am (re)descoperit o problemă a cărui enunț este cât se poate de simplu, rezolvarea în schimb este foarte frumoasă.

Enunț

Se consideră un șir de n numere întregi, printre care există cel puțin un element pozitiv. Scrieți un program care determină *secvența având suma elementelor maximă.*

Exemplu

n=10
Șirul: 1 2 -6 3 4 5 -2 10 -5 -6
Rezultate:
Suma maximă: 20
Lungimea secvenței: 5
Poziția de început a secvenței: 4
Poziția de sfârșit a secvenței: 8
Secvența având suma 20: 3 4 5 -2 10

Varianta I

Cea mai simplă idee care conduce la o rezolvare a acestei probleme ar fi de a genera toate perechile de numere întregi st și dr având proprietatea: $1 \le st \le dr \le n$, calculând pentru fiecare pereche suma elementelor secvenței T[st..dr] și determinând secvența având suma maximă.

În acest caz algoritmul ar avea următorul nucleu:

```
Suma_Max:=T[1];
              { Suma_Max: suma maxima cautata }
for st:=1 to n do
       { st: pozitia de inceput a secventei }
  for dr:=st to n do
        { dr: pozitia de sfarsit a secventei }
  begin
    sum:=0;
             { sum: suma secventei actuale }
    for i:=st to dr do
      sum:=sum+T[I];
    if Suma_Max<sum
      then Suma_Max:=sum
  end;
```

Complexitatea acestui algoritm, format din trei cicluri imbricate este $O(n^3)$. Pentru $n = 1000$ un calculator cu performanțe medii va lucra aproape o oră. Această complexitate nu poate fi acceptată, de aceea vom căuta modalități de micșorare a complexității, astfel încât să avem mai întâi un algoritm pătratic, apoi unul liniar.

Varianta II

Căutând optimizări posibile, observăm că suma secvenței T[st..dr] poate fi determinată în funcție de suma calculată anterior, pentru secvența T[st..dr-1]. Evident, orice sumă T[st..dr] poate fi comparată cu suma maximă obținută până în acel moment, imediat după ce a fost calculată. Rezultă un nucleu de complexitate $O(n^2)$, deoarece în algoritm vom avea doar două cicluri imbricate:

```
Suma_Max:=T[1];
for st:=1 to n do
begin
  sum:=0;
  for dr:=st to n do
  begin
    sum:=sum+T[dr];
    if Suma_Max<sum then Suma_Max:=sum
  end
end;
```

Varianta III

Al doilea algoritm pătratic are la bază o idee nouă, dar care folosește spațiu suplimentar, dublând necesarul de memorie. În rezolvarea pe care o propunem în această variantă, presupunem că dimensiunea șirului nu va fi așa de mare încât acest inconvenient să facă imposibilă obținerea soluției pe această cale.

Vom nota cu `Sum_Part[i]` suma elementelor secvenței `T[1..i]`. Vom calcula aceste sume parțiale pentru valorile `i=0,1,...,n` la începutul algoritmului și le vom memora în tabloul `SumPart[0..n]`. În consecință, suma elementelor unei secvențe `T[st..dr]` se calculează simplu, folosind elementele acestui tablou auxiliar:

`T[st..dr]=Sum_Part[dr]-Sum_Part[st-1].`

Programul în care s-a implementat algoritmul care are la bază ideea prezentată, este următorul:

Listing VAR3.PAS

```
Program Secventa_de_suma_maxima;
const n=100;
var T:array[1..n] of Integer;  { sirul dat }
    Sum_Part:array[0..n] of Integer;
                  { sirul sumelor partiale }
    inceput,sfarsit,st,dr,i:Byte;
    Suma_Max,sum:Integer;
Begin
  Writeln('Sirul dat: ');
  Sum_Part[0]:=0;
  { este nevoie de Sum_Part[i-1] cand i=1 }
  for i:=1 to n do
          { in paralel cu citirea sirului }
          { se calculeaza sumele partiale }
  begin
    Read(T[i]);
    Sum_Part[i]:=Sum_Part[i-1]+T[i]
  end;
  Writeln;
  Suma_Max:=T[1];
     { initializarea sumei maxime cautate }
  for st:=1 to n do
    for dr:=st to n do
    begin
      sum:=Sum_Part[dr]-Sum_Part[st-1];
                          { sume curente }
      if Suma_Max<sum then
      begin
        Suma_Max:=sum;
                  { suma maxima curenta }
        inceput:=st;
         { pozitia de inceput a secventei }
        sfarsit:=dr
         { pozitia de sfarsit a secventei }
      end
    end;
  Writeln('Secventa avand suma maxima: ');
  for i:=inceput to sfarsit do
    Write(T[i],' ');
  Writeln
End.
```

Evident, există și alte posibilități de a rezolva această problemă în complexitate pătratică; propunem cititorilor să caute un algoritm recursiv, folosind observațiile prezentate.

Varianta IV

În continuare urmează descrierea unui algoritm având complexitatea $O(n)$. Această performanță se obține datorită faptului că tabloul se parcurge o singură dată.

În paralel cu parcurgerea șirului de la stânga la dreapta vom determina și memora la fiecare pas suma maximă posibilă, în variabila `Suma_Max`. Să presupunem că am aflat suma maximă și secvența `T[1..i-1]` respectiv. Se pune problema extinderii metodei pentru calcularea sumei maxime pentru `T[1..i]`.

Se observă că după parcurgerea primelor `i` elemente ale șirului, suma maximă este egală cu suma maximă a primelor `i-1` elemente (`Suma_Max`) sau este mai mare decât aceasta. Dacă este mai mare, înseamnă că din această sumă maximă face parte și elementul de pe poziția `i`, deci avem un subșir care are elementul de sfârșit cel de-al `i`-lea element din șirul dat (această sumă o vom numi `Suma_C`).

Suma `Suma_C` nu va fi calculată de fiecare dată de la început, ci vor fi aplicate raționamentele deja cunoscute: se va utiliza suma maximă a secvenței ce se termină pe poziția `i-1`. În consecință obținem:

```
Suma_Max:=T[1];
Suma_C:=Suma_Max;
for i:=1 to n do
begin
  Suma_C:=max(Suma_C+T[i],0);          {1}
  Suma_Max:=max(Suma_Max,Suma_C)       {2}
end;
```

La examinarea acestei secvențe o deosebită atenție merită acordată variabilei `Suma_C`. Înainte de a fi executată instrucțiunea de atribuire `{1}` din corpul ciclului, valoarea acestei variabile este egală cu suma maximă a elementelor secvenței care se termină în poziția `i-1`. În urma execuției acestei instrucțiuni de atribuire în `Suma_C` va fi depusă suma maximă a elementelor secvenței care se termină în poziția `i` și începe într-o poziție memorată în momentul începerii însumării elementelor secvenței începând cu poziția `poz_inceput`. Evident, la primul pas această variabilă are valoarea `1`, indicele primului element din șir.

Este momentul să ne aducem aminte că șirul dat conține *cel puțin un element de valoare pozitivă*, (proprietate neexploatată în nici una din variantele de rezolvare prezentate). Din această particularitate rezultă că dacă șirul ar avea `n-1` elemente negative și unul singur pozitiv, secven-

ța având suma maximă s-ar reduce la un singur element și anume la acest număr pozitiv care ar da naștere la o sumă pozitivă. Rezultă că suma maximă căutată în orice situație va fi o sumă pozitivă. În consecință, cât timp suma curentă este pozitivă, prin atribuirea {1}, lui Suma_C i se adaugă T[i]. Dacă, la un moment dat se constată că Suma_C este negativă, atunci acestei variabile i se atribuie valoarea elementului T[i].

În ambele cazuri variabila Suma_C va fi comparată cu Suma_Max, actualizând valoarea Suma_Max după caz. În continuare urmează varianta finală a programului *Pascal*.

Listing VAR4.PAS

```
Program Secventa_de_suma_maxima;
const n=100;
var T:array[1..n] of Integer;  { sirul dat }
    Suma_Max,          { suma maxima cautata }
    Suma_C:Integer;
                       { suma maxima curenta }
    inceput,
          { pozitia de inceput a secventei }
    sfarsit,
          { pozitia de sfarsit a secventei }
    poz_inceput,     { pozitia de inceput a }
                     {    secventei curente }
    i:Byte;
Begin
  Writeln('Sirul dat: ');
  for i:=1 to n do Read(T[i]);
  Writeln;
             { initializarea sumei maxime  }
  Suma_Max:=T[1];
             { initializarea sumei curente }
  Suma_C:=Suma_Max;
       { initializarea pozitiei de inceput }
       {              a secventei curente }
  poz_inceput:=1;
       { initializarea pozitiei de inceput }
       {       a secventei de suma maxima }
  inceput:=1;
       { initializarea pozitiei de sfarsit }
       {         a secventei de suma maxima }
  sfarsit:=1;
  for i:=2 to n do
  begin
    if Suma_C<0 then
              { daca Suma_C este negativa }
    begin
      { aceasta se reinitializeaza cu T[i] }
      Suma_C:=T[i];
       { noua secventa posibila incepe la }
       {                          pozitia i }
      poz_inceput:=i
    end
    else
              { lui Suma_C i se adauga T[i] }
      Suma_C:=Suma_C+T[i];
    if Suma_Max<Suma_C then
                { se actualizeaza Suma_Max }
    begin
      Suma_Max:=Suma_C;
       { actualizarea pozitiei de inceput }
      inceput:=poz_inceput;
       { actualizarea pozitiei de sfarsit }
      sfarsit:=i
    end
  end;
  Writeln('Suma maxima: ',Suma_Max);
  Writeln('Lungimea secventei: ',
                          sfarsit-inceput+1);
  Writeln('Incepe din pozitia: ',inceput);
  Writeln('Sfarsitul in pozitia: ',sfarsit);
  Writeln('Secventa avand suma maxima: ');
  for i:=inceput to sfarsit do
    Write(T[i],' ');
  Writeln
End.
```

În urma executării programului pentru șirul T=(4,2,-7,9,-10,3,10,-3,5,-15), rezultatele furnizate de program sunt:

```
Suma maxima: 15
Lungimea secventei: 4
Incepe din pozitia: 6
Sfarsitul in pozitia: 9
Secventa avand suma maxima: 3 10 -3 5
```

În urma executării programului pentru șirul T=(1,2,3,-1,-2,-3,6,7,-7,8), rezultatele furnizate sunt:

```
Suma maxima: 14
Lungimea secventei: 10
Incepe din pozitia: 1
Sfarsitul in pozitia: 10
Secventa avand suma maxima: 1 2 3 -1 -2 -3 6
                            7 -7 8
```

Se observă că secvența obținută ca rezultat are lungimea maximă printre secvențele care au aceeași sumă maximă. În acest exemplu secvența 6 7 -7 8 are, de asemenea, suma 14. Lăsăm pe seama cititorului să caute o soluție care să furnizeze ca rezultat secvența de lungime minimă și având suma maximă. De asemenea, se poate pune întrebarea: cum procedăm dacă ne interesează prima secvență având proprietatea cerută, ultima sau eventual, toate secvențele.

Bibliografie:

Gh. Bostan, I. Mocanu, *Al VII-lea concurs republican de informatică*, Chișinău, 1993, nepublicat

D-na lect. univ. Clara Ionescu este redactor-șef al GInfo. Poate fi contactată prin e-mail la adresa **clara@cs.ubbcluj.ro**.

Publicarea unui SITE Web

Marian Veţeanu

Evoluţia din ultimii ani a Internet-ului şi a Web-ului a determinat tot mai multe firme şi persoane particulare să-şi dorească propriul site Web care să poată fi accesat de oricine din lume. Dacă pentru o firmă problema se rezolvă destul de uşor, utilizatorii particulari sunt puşi în faţa unor dificultăţi când este vorba de realizarea şi publicarea pe Web a unei pagini HTML.

În momentul în care se ia decizia publicării unui *site Web*, apar o mulţime de întrebări de genul:

- Cu ce program pot să-mi creez propria pagină *Web*?
- Pe ce server să o găzduiesc?
- Ce trebuie să fac pentru ca lumea să-mi găsească *site*-ul?
- Mă costă ceva pentru a avea propriul *site Web*?
- Pot să ştiu câţi utilizatori îmi vizitează *site*-ul?

Acestea sunt doar o parte din întrebările pe care şi le pune orice începător în *Web* design. Articolul de faţă încearcă să ofere câteva răspunsuri generale la întrebările de mai sus, împreună cu o colecţie de adrese *Web* ale unor *site*-uri care oferă servicii sau documentaţii utile începătorilor.

Principalii paşi în publicarea unui *site Web* sunt următorii:

- Crearea *site*-ului *offline* sau *online*;
- Găzduirea lui pe un server *Internet*;
- Obţinerea unui nume pentru *site*-ul *Web*;
- Anunţarea *site*-ului;
- Urmărirea vizitatorilor.

Crearea site-ului offline sau online

Pentru realizarea manuală a *site*-ului tot ce vă trebuie este un simplu editor *ASCII*, un editor grafic, cunoştinţe de *HTML* şi, eventual, de limbaje de *scripting* client (*JavaScript*, *VBScript*) şi server (*Perl*, *PHP*, *ASP*). În general utilizatorii şi programatorii avansaţi optează pentru realizarea *site*-ului în acest mod. Deşi un simplu program de tip *Notepad* este suficient, există totuşi o serie de instrumente care permit editarea paginilor cu facilităţi suplimentare: *syntax highlight*, *cod completition* etc. Dintre programele de acest gen se remarca :

- *Microsoft Visual Interdev* (www.microsoft.com)
- *Homesite*

Site-ul poate fi creat şi vizual prin folosirea de programe *WYSIWYG* (*What You See Is What You Get - ceea ce vezi este ceea ce obţii*) care permit realizarea şi întreţinerea de *site*-uri *Web* fără ca utilizatorul să posede cunoştinţe de *HTML* sau programare. Începătorii vor alege cu siguranţă această metodă.

Dintre zecile de programe de acest gen vă recomandăm folosirea unuia dintre următoarele:

- *NetObjects Fusion* (www.netobjects.com)
- *Microsoft FrontPage* (www.microsoft.com)
- *Macromedia Dreamweaver* (www.macromedia.com)

Utilizatorii mai grăbiţi, care nu doresc să investească timp în învăţarea diferitelor programe, îşi pot crea *site*-ul *online*, folosind diverse servicii puse la dispoziţie, în general, de *site*-urile ce oferă găzduire *Web*.

Găzduirea site-ului pe un server Web

Publicarea pe *Web* a *site*-ului se poate face pe serverele proprii (ale firmei, şcolii, universităţii etc.) sau pe serverele unei alte firme care oferă servicii de *webhosting*.

Serviciile de *webhosting* oferite de firme pot fi cu plată (între 5$ şi 300$ pe lună, în funcţie de serviciile oferite) sau gratuite. În cazul celor gratuite, *site*-urile respective inserează de obicei reclame pe paginile utilizatorilor.

Alegerea unui bun *site* de *webhosting* poate fi decisivă pentru afacerea dumneavoastră. Iată câteva întrebări pe care trebuie să le puneţi administratorului serverului care vă va găzdui paginile:

- Cât timp este disponibil? (Un procent bun este peste 98%.)
- Suportă *scripting* pe partea de server? (Puneţi întrebarea aceasta doar dacă *site*-ul dumneavoastră foloseşte *script*-uri pe partea de server.)
- Suportă baze de date? (Puneţi întrebarea aceasta doar dacă *site*-ul dumneavoastră este proiectat cu baze de date. În caz afirmativ trebuie să verificaţi dacă suportă baza de date dorită.)
- Vă asigură acces la statisticile vizitatorilor *site*-ului?

Majoritatea *site*-urilor de *free webhosting* nu suportă *scripting* pe partea de server sau, dacă suportă, este foarte limitat. Dacă vă hotarâți pentru găzduirea *site*-ului pe un server de *free webhosting* iată o listă cu astfel de *site*-uri:

- **www.home.ro** (server românesc cu acces rapid din România. Suportă extensiile *FrontPage* pentru publicarea *site*-ului și *FTP Upload*; lasă de dorit din punct de vedere al administrării);
- **www.angelfire.com**;
- **www.geocities.com**;
- **www.homestead.com**;
- **www.theglobe.com**;
- **www.tripod.com**;
- **www.xoom.com**;
- **www.webjump.com** (oferă *CGI*-uri în *Perl*; principalul dezavantaj este că introduce destul de multă publicitate pe paginile găzduite gratuit);
- **www.hypermart.net** (oferă *CGI*-uri în *Perl*; acceptă publicarea prin mai multe metode);
- **www.ewebcity.com** (fără publicitate, oferă *ASP*-uri, baze de date *Access*; principalul dezavantaj îl constituie stabilitatea destul de redusă).

Puteți găsi mai multe astfel de *site*-uri folosind orice motor de căutare sau *webdirector*.

Obținerea unui nume de domeniu

Dacă doriți ca afacerea dumneavoastră să pară cât mai serioasă, trebuie să obțineți un nume de domeniu propriu. Statistic vorbind, utilizatorii care navighează pe *Web* preferă adrese de tipul **www.nume.com** sau **www.nume.ro** față de cele de tipul **www.nume.isp/useri/vm/id/home.html**. Pentru a obține un nume de domeniu propriu, vizitați *site*-ul **www.ici.ro** (pentru domenii **.ro**) sau **www.internic.net** (pentru domenii **.com**, **.net**, **.org**).

Popularizarea *site*-ului

Chiar dacă *site*-ul dumneavoastră este foarte atrăgător, acesta este aproape inutil dacă nu este cunoscut de cei cărora li se adresează. Sunt multe metode prin care vă puteți populariza *site*-ul:

- *offline*: prin reclame TV, radio, ziare, reviste, participarea la sesiuni și conferințe tematice etc.
- *online*: prin înscrierea la motoare de căutare și *webdirector*-i, folosirea unui program de schimb de *banner*-e publicitare, înscrierea la liste de discuții specifice profilului *site*-ului, postarea de mesaje pe diverse *site*-uri, participarea la diverse conferințe, scrierea de articole din domeniul pentru care este proiectat *site*-ul Web, oferirea vizitatorilor de diverse servicii gratuite care să-i atragă și să-i determine să revină ulterior.

Pe această pagină găsiți o listă cu motoare de căutare la care puteți să vă înscrieți *site*-ul *Web*. Căutați în paginile acestora o legătură de tipul `Add URL` sau `Adaugă site`. Lista cuprinde principalele motoare de căutare internaționale, dar și foarte mulți *webdirector*-i din România și Republica Moldova.

Pentru ca respectivele motoare să vă indexeze corect *site*-ul, este responsabilitatea dumneavoastră să respectați câteva reguli de bază în momentul în care îl proiectați și îl construiți:

- titlurile paginilor să reflecte clar conținutul și să specifice numele persoanei sau firmei;
- toate documentele *HTML* trebuie să conțină marcajul **<META>** împreună cu cele două atribute absolut necesare: **keywords** și **description**.

Pentru a înțelege mai bine modul de folosire al marcajului **<META>** vă prezentăm forma în care apare la **www.psihoteste.home.ro**:

```
<META name="Description"
      content="Psihoteste - Teste
      psihologice distractive">
<META name="Keywords" content="psihotest,
      test, psihoteste, funtest,
      psihologic, psycho, psychotest,
      psycho tests, distractie, fun,
      amuzant, vma, vmasoft, vma soft,
      free, gratuit, online">
```

Numai despre acest marcaj și modul lui de folosire există pe *Internet* sute de pagini. Vă recomandăm să consultați pe această temă următoarele documentații:

Motoare de căutare

- http://www.altavista.com
- http://www.yahoo.com
- http://www.lycos.com
- http://www.hotbot.com
- http://www.acasa.ro
- http://www.index.ro
- http://www.home.ro
- http://www.start.ro
- http://www.rol.ro
- http://www.cauta.ro
- http://www.gaseste.com
- http://www.excite.com
- http://www.kappa.ro
- http://www.orientation.ro
- http://www.romaniabynet.ro
- http://www.millennium.ro
- http://www.iromania.com
- http://www.top100.ro
- http://www.moldova.net/english/
- http://www.ournet.md

- *Effective Web Marketing* (http://www.aid4promotion.com/book);
- *E-Marketing* (http://www.resurse.com/emarketing/index.shtml);
- *Search Engine Success - Ten Steps* (articol la http://www.about.com);
- Articolele de pe *site*-ul *Add-me* (http://www.addme.com);
- *Yahoo secrets* (http://yahoosecrets.com/yahoo/);
- *How do I get awards for my web site* (http://www.wsabstract.com);
- *Promoting your pages* (http://osu.orst.edu/aw/promote/).

O altă metodă eficientă pentru a vă populariza *site*-ul este înscrierea acestuia în rețele de schimb de *banner*-e sau cumpărarea de spații publicitare în cadrul altor *site*-uri de profil. Iată câteva adrese ale unor *site*-uri care se ocupă cu schimbul de *banner*-e publicitare:

- http://www.goldenclick.ro (românesc - asigură o rată de 1:1);
- http://www.grafstat.ro (românesc - asigură rate de afișare de 1:2, 1:3);
- http://www.hyperbanner.com;
- http://www.linkexchange.com;
- http://www.smartage.com;
- http://www.bcentral.com.

Vă recomandăm să vă înscrieți *site*-ul într-o rețea românească, în cazul în care acesta are conținutul în limba română. Un utilizator din altă țară nu se va uita la *site*-ul dumneavoastră în cazul în care nu cunoaște limba română.

Urmărirea vizitatorilor *site*-ului

Foarte mulți utilizatori folosesc diverse contoare puse pe prima pagină pentru urmărirea vizitatorilor *site*-ului. Pe lângă faptul că majoritatea contoarelor sunt inestetice din punct de vedere grafic, acestea mai creează și o impresie defavorabilă vizitatorilor *site*-ului. Este ca și cum ați telefona la 931 și operatoarea v-ar răspunde cu un mesaj de genul "*Bună ziua! Sunteți al 13782392 utilizator al serviciului nostru! Vă rugăm...*" După cum probabil ați observat, pe paginile *site*-urilor importante nu apar astfel de contoare.

O mai bună metodă de urmărire a vizitatorilor constă în folosirea serviciilor oferite de server-ul care vă găzduiește paginile sau de cel care vă asigură schimbul de *banner*-e. Iată și două servicii românești care se ocupă cu efectuarea de statistici asupra vizitatorilor:

- www.trafic.ro
- www.grafstat.ro

Pentru utilizatorii avansați și pentru programatorii *Web* oferim în continuare o colecție de adrese referitoare la programarea *Web*:

- http://www.mysql.com - *site*-ul *MySQL*
- http://www.php.net - *site*-ul *PHP*
- http://www.phpbuilder.com - resurse pentru programatorii *PHP*
- http://php.thescripts.net - resurse pentru programatorii *PHP*
- http://px.sklar.com - *PHP Code Exchange*
- http://www.hotscripts.com - *script*-uri pentru *Web*: *PHP*, *Perl*, *JavaScript*, *ASP*, *VBScript* etc.
- http://vmasoft.hypermart.net - secțiunea *WebApplications* a *site*-ului conține resurse pentru programatorii *PHP*, *ASP*, *Delphi*
- http://www.devshed.com - tutoriale pentru script-uri *Web*
- http://www.webmonkey.com - tutoriale pentru *Web*
- http://www.javascript.com - *JavaScript*
- http://webreference.com - tehnologii *Web*: *DHTML*, *JavaScript* etc.
- http://www.internet.com - *site* central pentru o mulțime de alte *site*-uri dedicate diverselor tehnologii *Web*
- http://www.perl.org - *site* pentru programatorii și utilizatorii *Perl*
- http://developer.netscape.com - documentații și exemple pentru majoritatea tehnologiilor *Web*
- http://www.worldwidemart.com/scripts - *Matt's script archive* - *script*-uri *Perl* gratuite: *guestbook*, *contor* etc.
- http://javascript.internet.com - *Javascript*
- http://www.javascript.com - *Javascript*
- http://www.webreference.com - *site* pentru dezvoltatorii *Web*
- http://www.webdeveloper.com - *site* pentru dezvoltatorii *Web*
- http://wsabstract.com - documentații *Javascript* și alte tutoriale pentru *Webdevelopment*
- http://www.interbase2000.com - *Site*-ul *Interbase*
- http://www.wdvl.com - *site* pentru dezvoltatorii *Web* (inclusiv tutorialul *Javascript for Programmers*)
- http://www.4guysfromrolla.com - despre *ASP*
- http://www.w3.org/TR/DOM-Level-2/ - *Document Object Model (DOM) Level 2 Specification*
- http://www.15seconds.com - documentații și exemple de surse *ASP*
- http://www.devhead.com/filters/dhtml - *DHTML*
- http://www.dhtmlzone.com - *DHTML*
- http://www.asptoday.com - despre *ASP*
- http://www.learnasp.com - despre *ASP*
- http://www.dynamicdrive.com - *script*-uri *DHTML*
- http://www.xml101.com - tutoriale *XML*
- http://www.w3schools.com/ - tutoriale *ASP*, *PHP*, *XML*, *HTML*, *DHTML*, *XHTML*, *CSS* etc.
- http://www.asp101.com - despre *ASP*
- http://www.devasp.com/ - *Developers ASP*
- http://www.findtutorials.com - *webdirector* pentru *site*-uri cu tutoriale și documentații
- http://www.thescripts.com/ - *script*-uri *ASP*, *PHP*, *DHTML* etc.
- http://www.chilisoft.com - oferă *ASP* independent de platformă
- http://www.waplinks.com - *WAP Links*
- http://php4ro.datablocks.net/ - *site* pentru dezvoltatorii români de *PHP*

Domnul Marian Veteanu este colaborator al GInfo și poate fi contactat prin e-mail la adresa vmasoft@yahoo.com.

Geometrie 3D

Mihai Scorțaru

În acest articol vă vom prezenta câțiva algoritmi care necesită unele cunoștințe de geometrie în spațiu pentru a putea fi înțeleși. În cazul acestor algoritmi, geometria are un rol secundar, ei rezolvând, de fapt, probleme de analiză combinatorică.

Aplicația 1

Se dau patru puncte necoplanare. Să se determine numărul de plane care se află la distanță egală față de cele patru puncte.

Rezolvare

Deoarece cele patru puncte nu sunt coplanare, rezultă că nu există un plan egal depărtat de aceste puncte care să se afle de aceeași parte a acestora. Rezultă că mai avem de tratat două cazuri:

- un punct se află de o parte a planului și celelalte trei de cealaltă parte;
- de fiecare parte a planului se află câte două puncte.

În primul caz, presupunem că punctele A, B și C se află de o parte a planului P, iar punctul D se află de cealaltă parte. Punctele A, B și C nu pot fi coliniare pentru că, altfel, ar exista un plan care ar conține toate cele patru puncte. Rezultă că punctele A, B și C determină un plan P' unic și, deoarece distanța dintre punctele A, B, C și planul P trebuie să fie aceeași, mai rezultă că planul P' este paralel cu planul P. Pentru ca distanța de la punctul D la planul P să fie aceeași cu distanța de la A, B și C, planul P trebuie să treacă prin mijlocul perpendicularei coborâte din D pe planul P'. Astfel, am demonstrat că există un singur plan egal depărtat de punctele A, B, C și D care are punctele A, B și C de o parte a sa și punctul D de cealaltă parte. Analog se demonstrează existența a încă trei asemenea plane: unul care are de o parte punctele A, B și D, iar de cealaltă parte punctul C, unul care are de o parte punctele A, C și D, iar de cealaltă parte punctul B și unul care are de o parte punctele B, C și D, iar de cealaltă parte punctul A. Considerând piramida determinată de punctele A, B, C și D, cele patru plane determinate până acum sunt paralele cu fețele piramidei și trec prin mijloacele înălțimilor corespunzătoare acestor fețe.

În al doilea caz presupunem că punctele A și B se află de o parte a planului P, iar punctele C și D se află de cealaltă parte. Deoarece planul P trebuie să fie egal depărtat de punctele A și B, rezultă că el trebuie să fie paralel cu dreapta AB. În mod analog se demonstrează faptul că planul P trebuie să fie paralel cu dreapta CD. Deoarece A, B, C și D nu sunt coplanare, rezultă că dreptele AB și CD nu sunt paralele și nici concurente. Alegem două plane paralele care să treacă prin AB și CD. Este evident că aceste două plane sunt paralele cu planul P și trebuie să fie egal depărtate de el. Astfel am demonstrat că există un singur plan egal depărtat de punctele A, B, C și D care are punctele A și B de o parte și punctele C și D de cealaltă parte. Analog, se demonstrează existența a încă două asemenea plane: unul care are de o parte punctele A și C, iar de cealaltă parte punctele B și D și unul care are de o parte punctele A și D, iar de cealaltă parte punctele B și C. Considerând piramida determinată de punctele A, B, C și D, cele trei plane determinate acum sunt paralele cu o pereche de muchii opuse și sunt egal depărtate de acestea.

În concluzie, avem șapte plane care se află la distanță egală față de patru puncte necoplanare date.

Aplicația 2

Determinați numărul maxim de zone în care poate fi împărțit spațiul prin n plane.

Rezolvare

Numărul maxim de zone se obține atunci când oricare două dintre plane nu sunt paralele, oricare trei dintre ele nu au nici o dreaptă comună și oricare patru dintre ele nu au nici un punct comun.

Vom presupune că am dus k plane și am împărțit spațiul într-un număr maxim de zone prin această operație. Vom determina cu cât crește numărul de zone după ce considerăm și al $(k+1)$-lea plan. Acest plan se va intersecta după k drepte cu planele deja existente, deoarece nu există perechi de plane paralele și oricare trei plane nu au drepte comune. Nici o pereche din cele k drepte nu sunt paralele,

deoarece nu există nici un grup de trei plane care să fie paralele cu o aceeași dreaptă. Nici un grup de trei din cele k drepte nu se intersectează în același punct deoarece nici un grup de patru dintre planele considerate nu trec prin același punct. Vom demonstra acum că cele k drepte împart noul plan în maximum $(k^2+k+2)/2$ zone. Astfel problema s-a redus la una de geometrie plană și anume problema determinării numărului maxim de zone în care poate fi împărțit un plan prin n drepte.

Planul va fi împărțit într-un număr maxim de zone dacă nu există nici o pereche de drepte paralele și nici un grup de trei drepte care să aibă un punct comun.

Presupunem că au fost duse k drepte; vom determina cu cât crește numărul de zone după ce considerăm cea de-a $(k+1)$-a dreaptă. Această dreaptă se intersectează cu celelalte k drepte în k puncte, deci va fi împărțită în $k+1$ zone. În concluzie, noua dreaptă va trece prin $k+1$ zone ale planului. Ea va împărți fiecare din aceste zone în două, deci numărul de zone va crește cu $k+1$.

Notând cu P_k numărul de zone în care poate fi împărțit planul prin considerarea a k drepte, obținem $P_{k+1}=P_k+k+1$. Deoarece prima dreaptă împarte planul în două zone, avem $P_1=2$. Vom deduce că $P_n=2+2+3+4+...+n$. De aici obținem că $P_n=(1+2+3+...+n)+1=n\cdot(n+1)/2+1=(n^2+n+2)/2$.

Să revenim acum la problema în spațiu. Am demonstrat că al $(k+1)$-lea plan va împărți spațiul în $(k^2+k+2)/2$ zone. Astfel, al $(k+1)$-lea plan va intersecta $(k^2+k+2)/2$ zone ale spațiului, fiecare dintre aceste zone fiind împărțită în două, deci numărul de zone va crește cu $(k^2+k+2)/2$ după considerarea celui de-al $(k+1)$-lea plan.

Notând cu S_k numărul de zone în care poate fi împărțit spațiul prin k plane, obținem $S_{k+1}=S_k+(k^2+k+2)/2$. Deoarece primul plan împarte spațiul în două zone, vom avea $S_1=2$.

Vom deduce că

$$S_n=2+\sum_{k=1}^{n-1}\frac{k^2+k+2}{2}=2+\frac{1}{2}\sum_{k=1}^{n-1}k^2+\frac{1}{2}\sum_{k=1}^{n-1}k+n-1$$

Deoarece $\sum_{k=1}^{n-1}k^2=\frac{n\cdot(n-1)\cdot(2\cdot n-1)}{6}$ și $\sum_{k=1}^{n-1}k=\frac{n\cdot(n-1)}{2}$, obținem

$$S_n=2+\frac{n\cdot(n-1)\cdot(2\cdot n-1)}{12}+\frac{n\cdot(n-1)}{4}+n-1$$

$$S_n=2+\frac{(n-1)\cdot(2\cdot n^2-n+3\cdot n+12)}{12}=\frac{n^3+5\cdot n+6}{6}.$$

Ca urmare, numărul maxim de zone în care poate fi împărțit spațiul prin n plane este $\frac{n^3+5\cdot n+6}{6}$.

Observație

Problema se poate generaliza pentru un hiperspațiu cu n dimensiuni, rezolvarea bazându-se pe reducerea problemei n-dimensionale, la una $(n-1)$-dimensională. De exemplu, pentru hiperspațiul cu 4 dimensiuni, am putea defini problema împărțirii acestuia prin n spații tridimensionale.

Se observă destul de ușor că vom obține formula de recurență $H_{k+1}=H_k+(k^3+5\cdot k+6)/6$, unde $H_1=2$.

Vom obține

$$H_n=2+\sum_{k=1}^{n-1}\frac{k^3+5\cdot k+6}{6}=2+\frac{1}{6}\sum_{k=1}^{n-1}k^3+\frac{5}{6}\sum_{k=1}^{n-1}k+n-1.$$

Folosind formula $\sum_{k=1}^{n-1}k=\frac{n^2\cdot(n-1)^2}{4}$ obținem

$$H_n=2+\frac{n^2\cdot(n-1)^2}{24}+\frac{5\cdot n\cdot(n-1)}{12}+n-1.$$

De aici avem

$$H_n=\frac{n^4+2\cdot n^3+n^2+10\cdot n^2-10\cdot n+24\cdot n+24}{24}=\frac{n^4+2\cdot n^3+11\cdot n^2+14\cdot n+24}{24}.$$

Aplicația 3

Determinați numărul maxim de sfere care să fie tangente la toate cele patru plane ale unui tetraedru.

Rezolvare

Problema se reduce la a determina numărul de puncte egal depărtate de cele patru fețe ale tetraedrului. Aceste puncte vor fi centrele sferelor.

Locul geometric al punctelor egal depărtate de fețele unui diedru este un plan care trece prin muchia diedrului și care împarte unghiul diedru în două unghiuri egale. Acest plan poartă denumirea de ***plan bisector*** al diedrului.

Deoarece două plane care se intersectează formează două perechi de diedre opuse la vârf, rezultă că locul geometric al punctelor egal depărtate de două plane care se intersectează este format din două plane care trec prin dreapta lor de intersecție.

Locul geometric al punctelor egal depărtate de fețele unui triedru este dreapta de intersecție a planelor bisectoare ale celor trei diedre ale triedrului. Această dreaptă de intersecție poartă denumirea de ***bisectoare*** a triedrului.

Deoarece trei plane care se intersectează formează patru perechi de triedre opuse la vârf, rezultă că locul geometric al punctelor egal depărtate de trei plane este format din patru drepte care trec prin punctul lor de intersecție.

Să considerăm acum planele P_1, P_2, P_3 și P_4 ale fețelor unui tetraedru. Considerăm triedrul interior tetraedrului format din planele P_1, P_2 și P_3. Locul geometric al punctelor egal depărtate de aceste trei fețe este format din patru drepte d_1, d_2, d_3 și d_4. Locul geometric al punctelor egal depărtate de planele P_1 și P_4 este format din două plane S_1 și S_2. Este evident faptul că oricare dintre punctele de intersecție ale uneia dintre dreptele d_1, d_2, d_3, și d_4 cu unul dintre planurile S_1 și S_2 va fi egal depărtat de planele P_1, P_2, P_3 și P_4, și că nu mai există alte puncte cu această proprietate. Astfel, în general, se obțin opt puncte egal depărtate de fețele piramidei, deci avem opt sfere tangente la cele patru plane ale unui tetraedru.

Observație

Există cazuri în care unele dintre dreptele d_1, d_2, d_3, și d_4 sunt paralele cu unul dintre planele S_1 și S_2. În aceste cazuri numărul de sfere se reduce. De exemplu, în cazul unui tetraedru regulat avem doar cinci sfere tangente la cele patru fețe.

INTEL contraatacă

Mihai Scorțaru

În luna decembrie a anului trecut AMD a redus substanțial prețurile produselor sale. Replica celor de la Intel nu s-a lăsat mult așteptată și, pe data de 29 ianuarie 2001, a fost adoptată o nouă grilă de prețuri, unele procesoare ieftinindu-se cu peste 30%.

Noutăți

Nici *Intel*, nici *AMD*, nu au lansat pe piață produse noi pentru *desktop*-uri. Chiar dacă se zvonea că pe data de 15 ianuarie *AMD* va introduce un procesor *Athlon Thunderbird* cu frecvența de 1,3 GHz, acesta nu a apărut încă pe piață.

Cei de la *AMD* au afirmat că nu este nevoie de mărirea frecvenței procesorului *Athlon*, deoarece un *Athlon* la 1,2 GHz este superior unui *Pentium 4* la 1,5 GHz. Ei au mai spus că *Athlon* va atinge frecvența de 1,3 GHz atunci când Intel va lansa procesorul *Pentium 4* cu frecvența de 1,7 GHz. Probabil că cele două lansări vor avea loc în curând, dar încă nu e cunoscută data exactă a acestora.

Pe piața procesoarelor destinate calculatoarelor portabile situația este diferită. Atât *Intel*, cât și *AMD*, oferă două noi tipuri de astfel de procesoare. Au apărut primele procesoare *Mobile Duron*, dar și primele procesoare *Mobile Pentium III* care consumă foarte puțină energie. La acest capitol *Intel* stă mult mai bine decât *AMD*, deoarece aici nu contează prea mult viteza unui procesor, un rol mult mai important având-l consumul de energie.

În viitor *Intel* va lansa procesoare *Mobile Pentium III* cu frecvențe care vor atinge în curând bariera de 1 GHz, în timp ce *AMD* se va concentra asupra unui procesor care să consume cât mai puțină energie.

Prețurile Intel

Așa cum am spus, cel mai important eveniment al ultimei luni a fost reducerea de prețuri pentru procesoarele *Intel*. Scăderea prețurilor practicate de *Intel* pentru distribuitorii săi a dus la o scădere imediată și a prețurilor cu care persoanele fizice pot achiziționa procesoare.

Intel Celeron

La începutul lunii februarie prețurile procesoarelor *Celeron* erau următoarele:

Celeron 533	*55$*	*-8$*	*(-13%)*
Celeron 566	*55$*	*-3$*	*(-5%)*
Celeron 600	*61$*	*-1$*	*(-2%)*
Celeron 633	*60$*	*-3$*	*(-5%)*
Celeron 667	*71$*	*-3$*	*(-4%)*
Celeron 700	*78$*	*-1$*	*(-1%)*
Celeron 733	*85$*	*-27$*	*(-24%)*
Celeron 766	*112$*	*-41$*	*(-27%)*
Celeron 800	*138$*	*-32$*	*(-19%)*

Doar prețurile procesoarelor *Celeron* cu frecvențe de peste 700 MHz au scăzut semnificativ. O scădere mai mare a prețurilor celorlalte procesoare *Celeron* nu era posibilă datorită faptului că nu ar mai fi fost acoperite costurile de producție.

Intel Pentium III

Prețurile procesoarelor *Pentium III* cu magistrala de 133 Mhz sunt următoarele:

Pentium III 533	*101$*	*-13$*	*(-11%)*
Pentium III 600	*103$*	*-17$*	*(-14%)*
Pentium III 667	*114$*	*-15$*	*(-12%)*
Pentium III 733	*149$*	*+10$*	*(+7%)*
Pentium III 800	*171$*	*-3$*	*(-2%)*
Pentium III 866	*188$*	*-22$*	*(-10%)*
Pentium III 933	*224$*	*-65$*	*(-22%)*
Pentium III 1000	*259$*	*-145$*	*(-36%)*

Este pentru prima dată când un procesor *Pentium III* la 1 GHz devine mai ieftin decât un *Athlon* la 1,2 GHz. Totuși, la aceeași frecvență, procesoarele de la *AMD* sunt mai ieftine decât cele de la *Intel*.

La fel ca în cazul procesoarelor *Celeron*, și la *Pentium III* reducerile apar doar la procesoarele cu frecvențe mari (peste 800 MHz). Reducerea prețurilor procesoarelor cu

frecvențe mai mici ar fi dus la prețuri mai mici decât ale procesoarelor *Celeron* cu aceeași frecvență.

Intel Pentium 4

La aceste procesoare se observă cele mai mari reduceri. Prețurile sunt următoarele:

Pentium 4 1300	*343$*	*-56$*	*(-14%)*
Pentium 4 1400	*390$*	*-201$*	*(-34%)*
Pentium 4 1500	*598$*	*-235$*	*(-28%)*

Numărul mic al procesoarelor *Pentium 4* la 1,3 GHz disponibile pe piață a dus la o relativă menținere a prețului acestora. Chiar dacă *Intel* a redus și prețul acestui procesor, reducerea nu a fost observată încă în magazine datorită stocurilor limitate.

Prețurile AMD

Chiar și după ce *Intel* a redus prețurile, este mult mai convenabilă achiziționarea unui procesor *AMD*. Din aceste motive *AMD* nu a aplicat o nouă reducere de prețuri.

AMD Duron

În Statele Unite un procesor *AMD Duron* poate fi achiziționat la următoarele prețuri:

Duron 600	*47$*	*0$*	*(0%)*
Duron 650	*46$*	*+5$*	*(+12%)*
Duron 700	*48$*	*-3$*	*(-6%)*
Duron 750	*59$*	*-3$*	*(-5%)*
Duron 800	*68$*	*-5$*	*(-7%)*
Duron 850	*93$*	*-12$*	*(-11%)*

Procesorul *Duron 850* a coborât pentru prima dată sub bariera de 100$. Comparând prețurile procesoarelor *Duron* cu cele ale procesoarelor *Celeron* cu aceeași frecvență, observăm că la 600 MHz *AMD* oferă un procesor cu 23% mai ieftin, la 700 MHz procesorul *Duron* este mai ieftin cu 38%, iar la 800 MHz cu 51%.

Un *Duron 850* este mai ieftin chiar și decât un *Celeron 766*.

AMD Athlon

Prețurile procesoarelor *Athlon Thunderbird* au rămas și ele aproximativ aceleași.

Athlon 700	*78$*	*-8$*	*(-9%)*
Athlon 750	*72$*	*-18$*	*(-20%)*
Athlon 800	*99$*	*-2$*	*(-2%)*
Athlon 850	*102$*	*0$*	*(0%)*
Athlon 900	*122$*	*-8$*	*(-6%)*
Athlon 950	*149$*	*-7$*	*(-4%)*
Athlon 1000	*179$*	*+6$*	*(+3%)*
Athlon 1100	*214$*	*-3$*	*(-1%)*
Athlon 1200	*260$*	*-2$*	*(-1%)*

La 800 MHz un *Athlon* este cu 42% mai ieftin decât un *Pentium III*, iar la 1000 MHz diferența scade la 31%. Un *Athlon* 800 este mai ieftin cu 12% decât un *Celeron 766* și cu 28% decât un *Celeron 800*.

Recomandări

Pentru calculatoarele ieftine cea mai bună alegere în acest moment pare a fi *Duron 700*. Chiar și în România prețul acestui procesor nu depășește 2.000.000 lei. Pentru calculatoarele performante un *Athlon Thunderbird* la 1 GHz este cea mai bună soluție în acest moment. În România prețul unui astfel de procesor este de aproximativ 7.500.000 lei. Pentru cei care vor să riște un *overclocking* procesorul recomandat este *Celeron 600*. Acesta poate atinge, relativ ușor, frecvența de 1008 MHz dacă dispuneți de o placă de bază performantă. În România un *Celeron 600* costă aproximativ 2.500.000 lei.

Totuși, ultimele reduceri de prețuri nu și-au făcut încă efectul și în România, așa că poate ar fi bine să mai așteptați puțin.

Mihai Scorțaru este redactor-șef adjunct al GInfo. Poate fi contactat prin e-mail la adresa **skortzy@xnet.ro**.

20 de replici ale programatorilor...

20. Ce ciudat...
19. Nu a mai făcut așa ceva!
18. Ieri a funcționat!
17. Cum se poate așa ceva?
16. Trebuie să fie o problemă hardware!
15. Ce ai introdus greșit ca să îl faci să se blocheze?
14. E ceva în neregulă cu datele tale!
13. Nu m-am mai atins de acest modul de câteva săptămâni!
12. Probabil că ai versiunea greșită.
11. E doar o coincidență nefericită!
10. Nu pot testa chiar totul!
9. Aceasta nu poate fi sursa acelei probleme!
8. Funcționează, dar nu a fost încă testat!
7. Probabil că cineva mi-a modificat codul.
6. Ai verificat dacă nu cumva ai un virus?
5. Chiar dacă nu funcționează, cum ți se pare?
4. Nu poți folosi această versiune pe calculatorul tău!
3. De ce vrei să faci așa? Poți și altfel!
2. Unde erai când s-a blocat?
1. Pe calculatorul meu funcționează!!!

Structură și stil în PROGRAMARE

Bazil Pârv

Continuăm serialul despre stilul în programare cu prezentarea modului în care ar trebui să fie proiectați și implementați algoritmii. Vom descrie modul în care trebuie alese structurile de date, precum și structura generală a unui program.

Toată analiza efectuată în episodul anterior a luat în considerare o problemă *P* generală. Am obținut următoarele:

- patru versiuni finale ale algoritmului de rezolvare a problemei *P*, care nu depind direct de problemă: `1`, `1R1`, `1R2` și `2M`;
- doi subalgoritmi generali, `Continua` și `AfiseazaMeniu`, independenți de problema *P* și apelați în versiunile `1R2` și respectiv `2M`;
- patru subalgoritmi specifici problemei *P*: `CitesteDate`, `CitesteDate2`, `Calculeaza` și `AfiseazaRezultate`, apelați în algoritmii de rezolvare a problemei *P*.

Vom continua să ignorăm enunțul problemei *P*, concentrându-ne atenția pe proiectarea tipurilor de date necesare, stabilirea modului de apelare a subalgoritmilor și pe exprimarea completă a algoritmilor care nu depind de problemă. Pseudocodul folosit în episodul precedent va fi înlocuit de limbajul *Pascal*, fără a pierde nimic din expresivitate.

Înainte de a trece la proiectare și implementare, vom identifica încă un element specific problemei *P*, care nu a fost luat în considerare până acum: ***antetul*** acesteia.

Prin antet înțelegem o scurtă explicație asupra problemei respective, care este afișată de fiecare algoritm înainte de a trece la rezolvarea ei.

Antetului problemei îi va corespunde încă un subalgoritm, a cărui specificare este:

```
SubAlgoritmul Antet este:
  @Afișează un rând ce sintetizează enunțul problemei
SfSubAlgoritm { Antet }
```

Subprogramul care implementează acest algoritm va avea următorul antet:

```
procedure Antet;
```

Proiectarea tipurilor de date

Respectând aceeași manieră ordonată de abordare a rezolvării problemei, ne impunem să definim tipuri de date utilizator pentru fiecare dintre datele evidențiate în analiză. Folosirea acestora ne va permite să exprimăm complet algoritmii și subalgoritmii care sunt în formă finală.

Operațiile meniului

Utilizatorul are de ales între patru operații: `Citire`, `Calcul`, `Afisare` și `Terminare`. Acestea se implementează natural prin tipul de date enumerare:

```
type TipOperatie=
        (Terminare,Citire,Calcul,Afisare);
```

Cu această notație, antetul subprogramului `AfiseazaMeniu` (care implementează subalgoritmul cu același nume) este:

```
procedure AfiseazaMeniu(var op:TipOperatie);
```

Datele de intrare ale problemei

Deoarece nu cunoaștem enunțul problemei, cea mai generală declarare a tipului datelor de intrare este cea de înregistrare:

```
type TipDate = record
    ...   { aici urmeaza componentele }
          { specifice datelor de intrare }
                { ale problemei P  }
 end;
```

Să remarcăm că pentru ce ne-am propus în această secțiune nu avem nevoie decât de numele tipului de date `TipDate`.

Antetele subprogramelor `CitesteDate` și `CitesteDate2` (care implementează algoritmii cu același nume) sunt:

```
procedure CitesteDate(var DI:TipDate);
procedure CitesteDate2(var DI:TipDate;
                       var ExistaDate:Boolean);
```

Rezultatele problemei

Declararea tipului rezultatelor este asemănătoare declarării tipului datelor de intrare:

```
type TipRezultate = record
     ...   {    aici urmeaza componentele }
           {       specifice rezultatelor }
           {                  problemei P  }
end;
```

Antetele subprogramelor `Calculeaza` și `AfiseazaRezultate` (care implementează algoritmii cu același nume) sunt:

```
procedure Calculeaza(DI:TipDate;
                     var RC:TipRezultate);
procedure AfiseazaRezultate(DI:TipDate;
                            RC:TipRezultate);
```

Proiectarea și implementarea subalgoritmilor generali

În această secțiune vom prezenta implementarea subalgoritmilor generali `Continuare` și `AfiseazaMeniu`.

Subalgoritmul `Continuare`

Acest subalgoritm este independent de problema *P*. Implementarea în *Pascal* este dată în continuare:

```
function Continuare:Boolean;
      { Pune intrebarea Continuati (D/N)? }
      { si accepta numai una din literele }
      {          'D', 'N', 'd', 'n' pentru }
      {                 variabila "raspuns". }
      {    Returneaza true daca "raspuns" }
      {         este 'D' sau 'd' si false }
      {                    in caz contrar }
var raspuns:Char;
            { caracterul care se citeste }
begin
  repeat
     { ne asiguram ca se citeste ce vrem }
    Write('Continuati (D/N)? ');
    Readln(raspuns);
    if not (raspuns in ['D','N','d','n'])
      then Writeln('Nu am inteles!')
  until raspuns in ['D','N','d','n'];
  Continuare:=raspuns in ['D', 'd']
end; { Continuare }
```

Față de descrierea prezentată în cadrul primului episod, implementarea conține verificarea strictă a caracterului citit în variabila `raspuns` (repetând cererea de citire la răspuns incorect) și ignoră diferența dintre literele mari și mici.

Subalgoritmul `AfiseazaMeniu`

Conform descrierii prezentate în primul episod, am păstrat acest algoritm independent de problema *P*. Implementarea sa este prezentată în continuare.

```
procedure AfiseazaMeniu(var op:TipOperatie);
  {  Afiseaza pe ecran meniul programului }
  { si citeste optiunea utilizatorului pe }
  {      care o salveaza in parametrul de }
  {                          iesire "op"  }
var o:Char;
begin
  Clrscr;
  repeat
    Writeln('Operatiile posibile sunt: ');
    Writeln('1. Citirea datelor de ',
                                 'intrare');
    Writeln('2. Calcularea rezultatelor');
    Writeln('3. Afisarea rezultatelor');
    Writeln('0. Terminarea executiei ',
                              'programului');
    Write('   Introduceti optiunea ',
                                  '(0-3): ');
      { Citeste o tasta fara a afisa pe }
      {  ecran caracterul corespunzator }
     o:=Readkey
  until o in ['0'..'3'];
    { conversie explicita la TipOperatie }
  op:=TipOperatie(Ord(o)-48);
        { trece la linie noua pe ecran }
  Writeln(Chr(10),Chr(13))
end; { AfiseazaMeniu }
```

Această implementare folosește două subprograme din unit-ul `Crt`: `Clrscr` și `Readkey`. Citirea codului operației dorite se face într-o variabilă caracter, citirea repetându-se dacă nu s-a introdus o valoare corectă. Tasta citită nu este afișată pe ecran, iar operația `op` se obține prin conversia explicită a caracterului citit la tipul `TipOperatie`.

Algoritmi pentru rezolvarea problemei P

În cele ce urmează este prezentată în întregime structura programelor. Textele sursă conțin numai declarațiile de variabile globale și corpurile programelor principale, în cele patru versiuni discutate, locul celorlalte declarații necesare fiind comentat.

Varianta 1

```
Program V1;
      { executie cu un singur set de date }
 { declaratiile tipurilor de date TipDate  }
 { si TipRezultate                         }
 { declaratia subprogramului Antet         }
 { declaratia subprogramului CitesteDate   }
 { declaratia subprogramului Calculeaza    }
```

```
  { declaratia subprogramului                    }
  { AfiseazaRezultate                            }
          { declaratia variabilelor globale }
var D:TipDate;          { datele de intrare }
    R:TipRezultate;          { rezultatele }
Begin
  Antet;          { subprogram de prezentare }
  CitesteDate(D);
          { subprogram care citeste datele }
  Calculeaza(D,R);
        { subprogram care face prelucrarile }
  AfiseazaRezultate(D,R);
     { subprogram care afiseaza rezultatele }
  Readln { pentru a putea studia rezultatele }
End. { V1 }
```

Varianta 2 (1R1)

```
Program V2;
      { executie cu seturi repetate de date }
      { datele de intrare dicteaza daca se  }
      { continua sau se opreste executia    }
  { declaratiile tipurilor de date TipDate  }
  { si TipRezultate                         }
  { declaratia subprogramului Antet         }
  { declaratia subprogramului CitesteDate2  }
  { declaratia subprogramului Calculeaza    }
  { declaratia subprogramului               }
  { AfiseazaRezultate                       }
          { declaratia variabilelor globale }
var D:TipDate;          { datele de intrare }
    R:TipRezultate;          { rezultatele }
    SuntDate:Boolean;
      { variabila booleana: true sau false }
Begin
  repeat
    Antet;          { subprogram de prezentare }
    CitesteDate2(D, SuntDate);
          { subprogram care citeste datele }
    if SuntDate then
    begin
      Calculeaza(D,R);
        { subprogram care face prelucrarile }
      AfiseazaRezultate(D,R);
                { subprogram care afiseaza }
                {              rezultatele }
      Readln
        { pentru a putea studia rezultatele }
    end
  until not SuntDate
End. { V2 }
```

Varianta 3 (1R2)

```
Program V3;
  { Executie cu seturi repetate de date.    }
  { Continuarea sau terminarea executiei    }
  { separata de citirea datelor de intrare. }
  { declaratiile tipurilor de date TipDate  }
  { si TipRezultate                         }
  { declaratia subprogramului Antet         }
  { declaratia subprogramului CitesteDate2  }
  { declaratia subprogramului Calculeaza    }
  { declaratia subprogramului               }
  { AfiseazaRezultate                       }
          { declaratia variabilelor globale }
var D:TipDate;          { datele de intrare }
    R:TipRezultate;          { rezultatele }
Begin
  repeat
    Antet;          { subprogram de prezentare }
    CitesteDate2(D);
          { subprogram care citeste datele }
    Calculeaza(D,R);
        { subprogram care face prelucrarile }
    AfiseazaRezultate(D,R);
                    {      subprogram care }
                    { afiseaza rezultatele }
  until not Continuare
End. { V3 }
```

Varianta 4 (2M)

```
Program V4;
  { executie cu seturi repetate de date,    }
  { meniu }
uses Crt;
          { se apeleaza Clrscr si Readkey }
          {              in AfiseazaMeniu }
  { declaratiile tipurilor de date TipDate  }
  { si TipRezultate                         }
  { declaratia tipului de date TipOperatie  }
  { declaratia subprogramului Antet         }
  { declaratia subprogramului CitesteDate2  }
  { declaratia subprogramului Calculeaza    }
  { declaratia subprogramului               }
  { AfiseazaRezultate                       }
  { declaratia subprogramului Mesaj         }
          { declaratia variabilelor globale }
var D: TipDate;          { datele de intrare }
    R: TipRezultate;          { rezultatele }
    operatie:TipOperatie;
                          { operatia ceruta }
    ExistaDate:Boolean;
          { s-au citit datele de intrare? }
    ExistaRezultate:Boolean;
            { s-au calculat rezultatele? }
Begin
  ExistaDate:=false;
  ExistaRezultate:=false;
  repeat
          { afiseaza meniul pe ecran si }
          {              citeste operatia }
    Antet;
    AfiseazaMeniu(operatie);
```

```
    case operatie of
      Citire:
        begin
          CitesteDate(D); { citeste datele }
          ExistaDate:=true;
          ExistaRezultate:=false
        end;
      Calcul:
        if ExistaDate then
          begin
            Calculeaza(D,R);
                        { face prelucrarile }
            Mesaj('Rezultate calculate!');
            ExistaRezultate:=true
          end
        else
          Mesaj('Nu s-au citit '+
                      'datele de intrare!');
      Afisare:
        if ExistaRezultate then
          begin
            AfiseazaRezultate(D, R);
                    { afiseaza rezultatele }
            ReadLn
          end
        else Mesaj('Nu s-au calculat '+
                                'rezultatele!')
    end { case }
  until operatie = Terminare
End. { V4 }
```

Dacă vom compara această implementare cu versiunea 2M, vom constata că s-a prevăzut un mesaj și în situația calculării rezultatelor (utilizatorul este înștiințat că s-a efectuat operația cerută) și că toate mesajele se afișează prin apelul unei proceduri Mesaj al cărei text sursă este prezentat în continuare.

```
procedure Mesaj(m:string);
       { afiseaza mesajul 'm' si asteapta }
       {                  apasarea unei taste }
var c:Char;
begin
  Writeln(m);
  Write('Apasati o tasta...');
  c:=ReadKey
end; { Mesaj }
```

În numărul următor

Ultimul episod al serialului va conține o prezentare a metodelor de implementare a elementelor specifice unei anumite probleme pe care dorim să o rezolvăm.

Vom exemplifica aceste metode pentru o problemă simplă: rezolvarea ecuației de gradul I.

Dl. prof. univ. Bazil Pârv este cadru didactic la Universitatea "Babeș-Bolyai" din Cluj. Poate fi contactat prin e-mail la bparv@cs.ubbcluj.ro.

Noi companii aeriene

Iată ce s-ar întâmpla dacă sistemele de operare ar fi companii aeriene:

DOS AIR

Toți pasagerii vor merge pe pistă, vor împinge avionul până când se desprinde, vor sări în avion, apoi vor sări din nou afară când avionul va atinge din nou pământul, îl vor împinge din nou etc.

WINDOWS AIRLINES

Terminalele sunt foarte curate, personalul are o ținută impecabilă, piloții sunt foarte capabili, iar flota e imensă. Avionul ajunge cu o întârziere de șase luni și după decolare, la înălțimea de 10.000 de metri explodează fără nici un avertisment.

MAC AIRWAYS

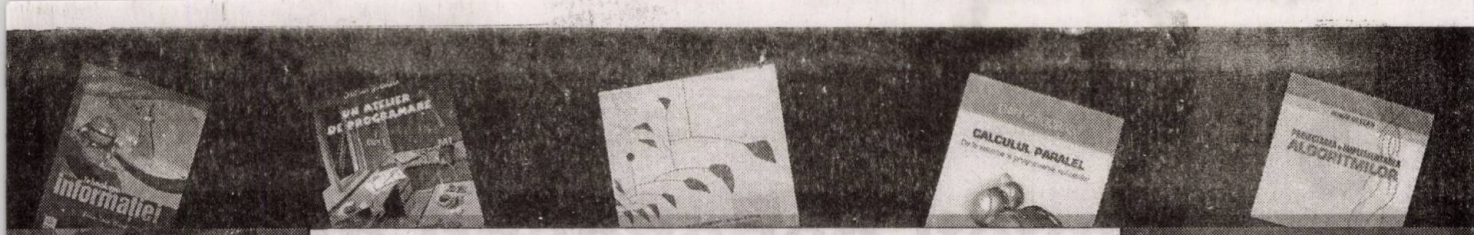
Casierii, personajul auxiliar și piloții arată la fel și acționează la fel. Dacă un pasager pune întrebări referitoare la zbor i se spune că nu vrea să știe, nu are nevoie să știe și este rugat să se întoarcă la locul său și să urmărească filmul.

OS/2 SKYWAYS

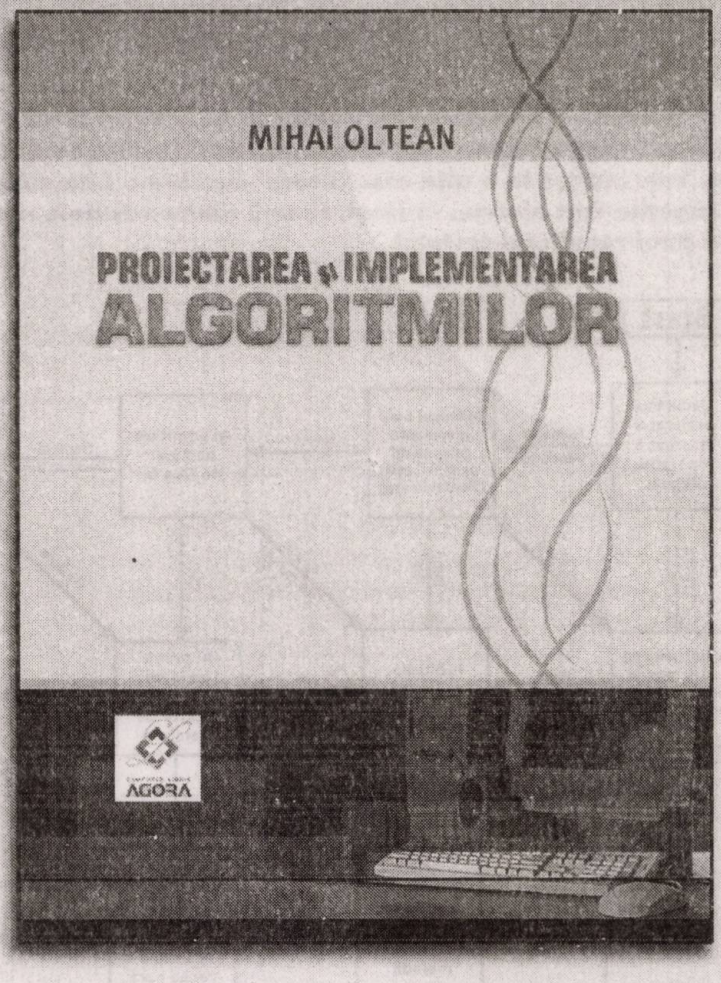
Terminalul este aproape gol, doar câțiva potențiali pasageri învârtindu-se prin zonă. Personalul le adresează permanent scuze arătându-le avionale de pe pistă. Fiecare potențial pasager află cât de confortabil va fi zborul cu aceste avioane, care sunt motivele pentru care zborul este mai sigur decât în cazul companiei Windows Airlines, dar este rugat să mai aștepte puțin deoarece tehnicienii mai au de lucru la sistemele de zbor.

LINUX EXPRESS

Fiecare pasager va aduce la aeroport o bucată din avion și o cutie cu unelte de lucru. Ei se întâlnesc pe pistă și se ceartă permanent deoarece nu au căzut încă de acord asupra tipului de avion pe care doresc să îl construiască și nici nu știu exact cum să îl asambleze. Până la urmă construiesc mai multe avioane, dar acestea primesc același nume. Unii pasageri chiar ajung la destinație, dar toți pasagerii, fără nici o excepție, sunt convinși că au ajuns unde trebuie.



COMPUTER LIBRIS AGORA



"Cum aș putea să justific existența acestei cărți? Cum aș putea spre exemplu să justfic cele 100 de pagini despre programarea dinamică, acum când această metodă nu mai figurează în programele școlare? Nu știu. Sau, cum aș putea să justific capitolul despre algoritmi genetici, când pe plan mondial abia de câțiva ani a fost dovedită utilitatea și eficiența lor? Din nou nu știu! Dar rămâne bănuiala: a scrie algoritmi înseamnă a crea. Și nu poți crea fără să gândești."

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

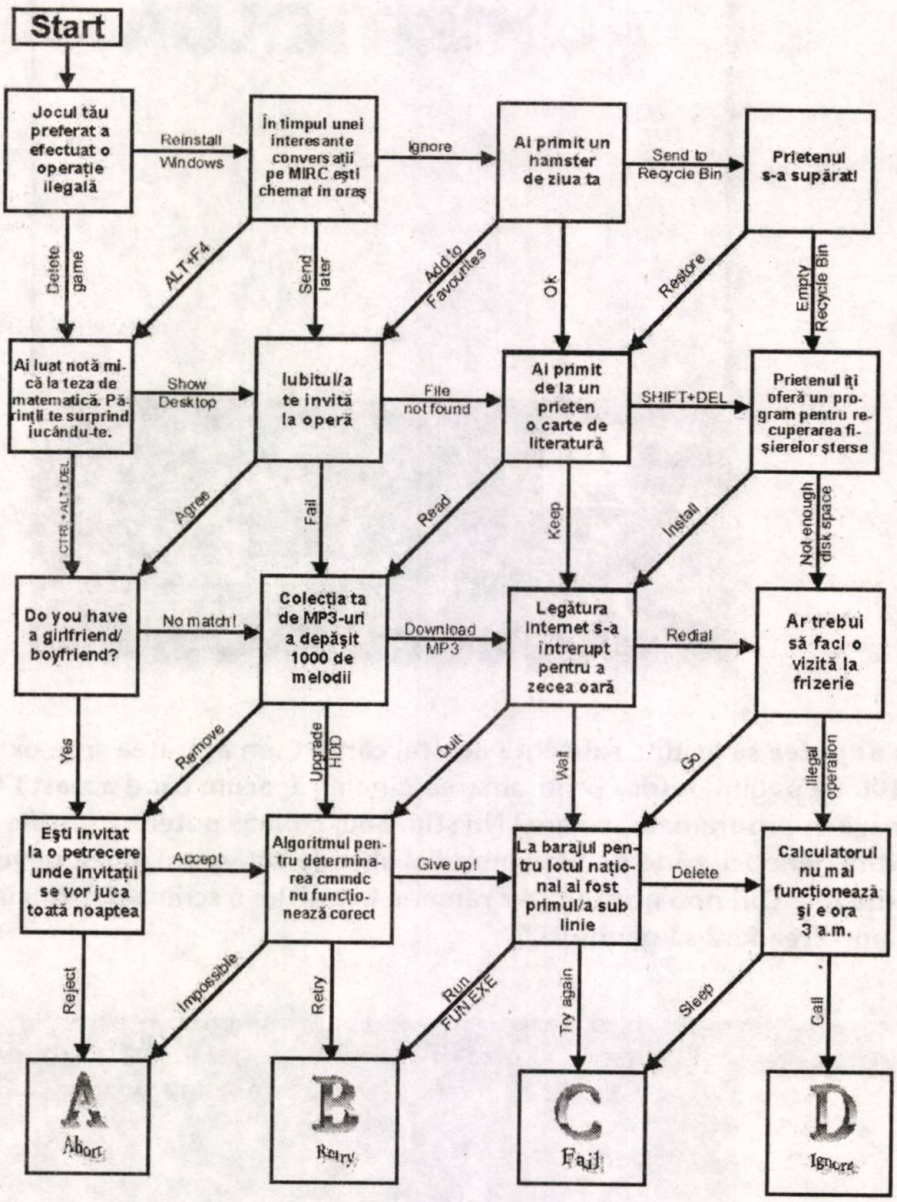
COMPUTER LIBRIS AGORA

Psychotest...

Sunteți dependenți de CALCULATOR?

Carmen Şuşman

Porniți de la START și imaginați-vă că sunteți în situația descrisă în prima căsuță. Alegeți săgeata corespunzătoare modului în care credeți că veți reacționa. Veți ajunge la o altă căsuță care prezintă o altă situație ipotetică. Urmând săgețile veți ajunge, în final, la una dintre căsuțele notate cu A, B, C și D. Citiți apoi rezultatul testului.



A. E bine că ignori calculatorul, dar nici chiar așa! Pentru tine calculatorul are o importanță atât de nesemnificativă încât, la apariția unei probleme (oricât de mică ar fi ea) ești gata să pleci din fața lui și preferi să-i lași pe alții să o rezolve. Mai grav este faptul că nici nu ești curios să arunci măcar o privire sumară peste soluție. Așa că data viitoare vei apela din nou la prieteni. Ar fi bine să-ți revizuiești comportamentul deoarece ești totuși cititor/cititoare al/a *GInfo*!

B. Fii fericit!!! Ai trecut testul cu bine. Ești un om normal căruia îi place viața. Știi să profiți de ocaziile care îți permit să te distanțezi de calculator și mai ales să le faci plăcute. Însă, atunci când trebuie, ești silitor/silitoare și conștiincios/conștiincioasă, dai dovadă de talent și de inteligență pentru că lucrurile serioase pentru care îți "sacrifici" timpul din plăcere îți ies întotdeauna. Ești o persoană destul de invidiată pentru modul tău de viață, adică îmbini utilul cu plăcutul, dar ai și prieteni adevărați de care ești mulțumit/mulțumită.

C. Nu știi să profiți de timpul tău prețios decât într-un singur mod: stând în fața calculatorului. Asta nu e foarte bine. Ar trebui să te destinzi și tu în diferite moduri: să citești o carte, să vizionezi un film, dar nu cu *Windows Media Player*, să mergi la o petrecere sau pur și simplu să nu-ți mai refuzi prietenii când te invită seara în oraș. Trebuie să remarci și persoanele de sex opus din anturajul tău pentru că, odată și odată, tot vei dori să-ți găsești jumătatea. Merită să încerci!

D. E grav! E foarte grav! Verdict: ***ești obsedat/obsedată de calculator!*** Dacă te plictisești de *Quake*, *Starcraft*, *Diablo*, *Need for Speed*, *Fifa 2000*, *Heroes* etc. soluția nu este să accesezi site-ul preferat, ci să încerci să faci alte lucruri mai interesante, care să te ajute să uiți măcar un pic de calculator, dacă nu chiar doi "pici". Fii sigur/sigură că vei reuși. Trebuie doar să-ți lași imaginația să acționeze sau pur și simplu să răspunzi la telefoanele prietenilor tăi, care sunt atât de nerăbdători să ieși cu ei în oraș. Nopțile sunt făcute a dormi sau pentru a te distra. Dacă se întâmplă lucruri grave, se înțelege că trebuie să dai telefoane la Poliție, Pompieri sau Salvare, dar să-ți repare cineva calculatorul la o oră atât de târzie... mai greu.

UPGRADE cu probleme

Plângerea

Anul trecut am făcut un upgrade de la *Boyfriend 5.0* la *Husband 1.0* și am observat că noul program efectuează schimbări neașteptate asupra celorlalte produse, limitând sever accesul la aplicațiile pentru garderobă, flori și bijuterii, care funcționau impecabil împreună cu *Boyfriend 5.0*.

Nici o mențiune referitoare la acest fenomen nu apărea în documentație. Mai mult, *Husband 1.0* a dezinstalat numeroase programe valoroase cum ar fi *DinnerDancing 7.5*, *CruiseShip 2.3* sau *OperaNight 6.1* și a instalat altele nedorite cum ar fi *PokerNight 1.3*, *SaturdayFootball 5.0*, *Golf 2.4* și *ClutterEverywhere 4.5*. *Conversation 8.0* nu mai funcționează și blochează sistemul. Nu mai rulează nici *DiaperChanging 14.1* sau *HouseCleaning 2.6*.

Am încercat să rulez *Nagging 5.3* pentru a repara *Husband 1.0*, dar efectul acestui utilitar este de scurtă durată.

Răspunsul

Aceasta este o problemă de care se plâng foarte multe femei după ce instalează *Husband 1.0*, dar se datorează unei concepții greșite. Multe femei fac *upgrade* de la *Boyfriend 5.0* la *Husband 1.0* fără să își de-a seama că *Boyfriend 5.0* este, în general, un program pentru distracție.

Husband 1.0 este un sistem de operare și a fost astfel proiectat încât să ruleze cât mai puține aplicații. Mai mult, nu puteți dezinstala *Husband 1.0* pentru a reîncepe utilizarea programului *Boyfriend 5.0*, deoarece *Husband 1.0* nu a fost proiectat astfel încât să permită acest lucru. Fișiere ascunse din sistemul dumneavoastră ar determina programul *Boyfriend 5.0* să emuleze *Husband 1.0*, deci nu ați avea nimic de câștigat. Este imposibil să dezinstalați sau să ștergeți *Husband 1.0* după ce l-ați instalat.

Deoarece am instalat și eu *Husband 1.0*, vă sugerez să citiți întreaga secțiune referitoare la *GPF* (*General Partnership Faults*). Aceasta este o facilitate extraordinară pe care compania producătoare a adăugat-o în secret la sistemul de operare. *Husband 1.0* trebuie să își asume întreaga responsabilitate pentru toate greșelile și problemele, indiferent de cine le creează. Pentru a activa această facilitate introduceți comanda `"C:\I THOUGHT YOU LOVED ME"`. Uneori *Tears 6.2* trebuie să ruleze în timp ce dați această comandă. Imediat *Husband 1.0* va lansa aplicațiile *Apologize 12.3* și *Flowers/Chocolates 7.8*.

Ar trebui să mai achiziționați câteva produse noi pentru a îmbunătăți performanța. Personal, vă recomand *HotFood 3.0*, *Lingerie 5.3* and *Patience 10.1*. Folosite împreună aceste utilitare vor face în așa fel încât *Husband 1.0* să ruleze fără nici o problemă.

După câțiva ani de folosință, *Husband 1.0* vă va deveni familiar și veți găsi numeroase facilități folositoare cum ar fi *FixesBrokenThings 2.1*, *Snuggling 4.2* sau *BestFriend 7.6*.

În final, vă avertizez să nu instalați în nici un caz *MotherInLaw 1.0*, deoarece aceasta nu este o aplicație suportată de acest sistem de operare și va determina căderi repetate ale acestuia.

Succes în utilizarea produsului *Husband 1.0*!

Warcraft III

Carmen Şuşman

Mulţi consideră că Warcraft III este cel mai aşteptat joc al anului 2001. Deşi acest joc a fost anunţat încă din 1999, data oficială a lansării nu a fost comunicată mult timp. Se pare că, totuşi, pasionaţii de jocuri pe calculator vor putea începe cea de-a treia aventură Warcraft începând cu al treilea trimestru al acestui an.

Despre Blizzard

Blizzard este unul dintre cei mai cunoscuţi producători de jocuri din întreaga lume. Chiar dacă numărul produselor lansate de această companie este relativ mic faţă de cel al rivalilor, fiecare titlu al său a fost un real succes.

Blizzard a revoluţionat jocurile de strategie în timp real, impunând standarde care trebuiau respectate de companiile rivale. Prima versiune a jocului *Warcraft* era practic o clonă a jocului *Dune II*, primul joc de strategie în timp real care a apărut pe piaţă. A doua versiune a fost mult mai bine primită de public, mulţi pasionaţi pierzând multe nopţi în bătăliile împotriva orcilor sau a oamenilor.

După succesul extraordinar al lui *Warcraft II*, *Blizzard* a continuat seria jocurilor de strategie cu *Starcraft*. Universul mitic al orcilor a fost înlocuit cu unul futurist în care trei rase luptau pentru controlul asupra galaxiei. Chiar dacă *Starcraft* a fost cel mai bun joc de strategie lansat de această firmă, mulţi pasionaţi regretau că orcii au dispărut de pe scenă. Pentru a-şi mulţumi fanii, *Blizzard* a considerat că este indicat să continue seria *Warcraft* şi aşa a început lucrul la cea de-a treia versiune.

Diablo a fost un alt joc de succes al companiei. Acesta era un *role-playing game* şi, cu ajutorul său, *Blizzard* a reuşit să impună standarde şi în acest domeniu. Firesc, în anul 2000 a apărut continuarea *Diablo II*, care a fost un real succes. *Diablo II* a fost considerat cel mai bun *role-playing game* al anului 2000.

Istoria jocului Warcraft III

Spre bucuria multor fani, la un moment dat, *Blizzard* a anunţat că a început să lucreze la cel de-al treilea joc din seria *Warcraft*. La început ni se promitea că vom putea alege să luptăm de partea uneia dintre cele şase rase care vor popula universul jocului. Iniţial au fost anunţate trei dintre rase: orcii şi oamenii erau deja cunoscuţi din jocurile anterioare, iar demonii au constituit prima noutate pe care au anunţat-o producătorii.

După un timp, *Blizzard* a anunţat că numărul raselor va fi redus la cinci şi, pentru a mai reduce tristeţea fanilor, a dat publicităţii şi o descriere a unei a patra rase: *undead*. Aceştia sunt scheletele morţilor readuse la viaţă prin intermediul magiei. Compania a mai afirmat că denumirea celei de-a cincia rase va fi anunţată la începutul anului 2001.

Luna ianuarie a acestui an a adus o nouă surpriză neplăcută pentru cei care aşteptau cu sufletul la gură să afle noutăţi despre *Warcraft III*. A fost într-adevăr anunţată o nouă rasă şi anume elfii întunericului (*Night Elves*). Din nefericire, numărul raselor s-a redus la patru, demonii fiind eliminaţi.

Blizzard a promis că cele patru rase actuale vor apărea în versiunea finală a jocului, dar ne putem aştepta la orice acum. Unii fani au spus că, în acest ritm, vom ajunge să rămânem doar cu cele două rase iniţiale: orcii şi oamenii.

Un nou gen

Chiar dacă primele jocuri din această serie erau strategii în timp real (***RTS*** - *Real Time Strategy*), *Blizzard* va încerca să impună un nou tip de joc prin intermediul lui *Warcraft III*. Compania a numit acest nou gen ***RPS*** (*Role-Playing Strategy*).

Acest nou concept doreşte să introducă jocurile de strategie într-un mediu *role-playing*. *RPS*-urile vor fi jocuri în care lupta individuală din ***RPG***-uri (*Role-Playing Games*) va fi combinată cu strategia din *RTS*-uri.

În *Warcraft III* jucătorii vor controla armate mai puţin numeroase, dar mult mai puternice. Ei se vor aventura într-o lume dinamică în care vor apărea clădiri şi monştri neutri, precum şi diferite alte personaje cu roluri bine de-

terminate. De asemenea, jucătorii trebuie să îndeplinească diferite sarcini pentru care vor fi recompensați. Va apărea un nou tip de unități și anume eroii care vor avea un rol asemănător cu cel din seria de succes *Heroes of Might and Magic*.

Firul epic

În orice joc, un rol foarte important îl are firul epic, adică povestirea care stă la baza acțiunii jocului.

Jocul *Warcraft II* se termina prin distrugerea portalului (*Dark Portal*) care permitea accesul între *Azeroth* (tărâmul oamenilor) și *Draenor* (tărâmul orcilor). Acțiunea din *Warcraft III* începe după o scurtă perioadă de timp de la acest eveniment.

Din nefericire, după ce au scăpat de orci, oamenii nu au urmat drumul onoarei, ci pe cel al înșelăciunii. Puținii orci rămași în viață au devenit sclavii oamenilor.

Unul dintre acești orici este *Thrall*, eroul principal al acțiunii din *Warcraft III*. Folosind calitățile învățate de la stăpânii săi, împreună cu puterea și curajul său natural, *Thrall* reușește să scape de sub stăpânirea oamenilor decăzuți și încearcă să își găsească moștenirea pierdută.

El a observat că semenii săi au fost forțați să ducă o viață pasivă în rezervații speciale. Totuși, unii au reușit să scape de furia *Alianței* și să se refugieze în sălbăticie.

Ajuns în aceste tabere ale orcilor, *Thrall* jură să își conducă frații în luptă pentru a ajunge din nou la statutul pe care îl cerea moștenirea lor.

Sub conducerea lui *Thrall*, care a devenit noul șef al războinicilor orci, aceștia își redescoperă cultura șamanistică abandonată și își folosesc noua putere pentru a se revolta împotriva asupritorilor.

Între națiunile *Alianței* izbucnește un război civil care duce la schimbări ciudate în *Azeroth*. În acest timp apare pericolul unui nou dușman, nevăzut: *Legiunea de Foc*. Până la urmă, *Legiunea* apare într-o ploaie de foc care cade din ceruri.

Membrii *Legiunii* au fost blestemați să cutreiere prin diferite ținuturi și să consume energiile primare ale magiei pentru a supraviețui. Ei au distrus nenumărate lumi pentru a putea să obțină energia necesară.

Scopul *Legiunii* nu era doar de a prelua toate energiile magice din ținut, ci și de a eradica orice formă de viață de aici.

S-ar părea că acum, în fața pericolului acestei invazii, *Hoarda* reunită a orcilor și armatele din *Azeroth* nu se vor mai concentra asupra cuceririlor, trădărilor, apărării sau onoarei, ci doar asupra supraviețuirii.

Din acest moment jucătorul are rolul de a influența evenimentele viitoare.

Aceasta este versiunea oficială, prezentată la mijlocul lunii februarie, a firului epic care stă la baza acțiunii din *Warcraft III*. Totuși, *Legiunea* era, de fapt, rasa demonilor; așadar s-ar putea ca acest fir epic să fie modificat, deoarece demonii au fost eliminați.

Informații generale

Chiar dacă *Blizzard* dorea să introducă conceptul de joc *RPS*, pe măsură ce jocul s-a dezvoltat partea de *RTS* a devenit mult mai importantă decât cea de *RPG*. Totuși, *Blizzard* susține că jocul va fi diferit față de *Warcraft II* sau *Starcraft*, dar, până la urmă, publicul va decide ce fel de joc este, după ce va fi lansat pe piață.

Jucătorii își vor construi clădirile într-un mod asemănător cu cel din *Starcraft*, însă nu se mai pune un accent la fel de mare pe aceste construcții și cei care se joacă vor construi mult mai puține clădiri.

Clădirile vor putea fi îmbunătățite pentru a produce unități mai avansate. De asemenea, vor putea fi dezvoltate noi capacități și magii.

Blizzard a promis că va exista și o versiune beta publică pentru *Warcraft III* deoarece astfel de teste s-au dovedit utile.

Pentru *Diablo II* a avut loc o lansare simultană la nivel mondial, jocul fiind disponibil în întreaga lume încă din prima zi. *Blizzard* a promis că va încerca o astfel de lansare și pentru *Warcraft III*.

Pentru *Diablo II* a apărut și o ediție pentru colecționari. *Blizzard* a promis că va exista și *Warcraft III Collector's Edition* pentru fanii acestei serii.

Va exista și o versiune pentru platforma *MacIntosh*, dar aceasta este așteptată doar spre sfârșitul acestui an. Din nefericire, nu este planificată și o versiune pentru *Linux* sau alte sisteme de operare. Momentan, firma se concentrează doar asupra versiunii *Windows*.

Rasele

Așa cum am mai spus, în *Warcraft III* vor fi patru rase: *Humans*, *Orcs*, *Undead* și *Night Elves*. Fiecare dintre aceste rase este de fapt o alianță între mai multe națiuni. De exemplu, rasa umană are, pe lângă unități aparținând oamenilor și altele, aparținând *dwarf*-ilor sau *elf*-ilor. Fiecare rasă va avea caracteristici proprii și va necesita o strategie diferită pentru a putea fi condusă.

Toate rasele vor avea aceeași importanță în timpul jocului. La fel ca și în *Starcraft*, oricare dintre rase este capabilă să obțină victoria. Totuși, în *Warcraft III*, fiecare rasă se va concentra asupra dezvoltării unor capacități proprii. Acesta este principalul motiv pentru care rasele vor adopta strategii diferite.

Jocul *multiplayer* este proiectat în așa fel încât să se bazeze pe echipe. Rasele sunt proiectate astfel încât să se completeze una pe alta în luptă, deci o echipă formată din două rase diferite va avea capacități unice pe care rasele luate individual nu le-ar avea.

Este foarte interesant modul în care se va dezvolta rasa *Undead*. Jucătorul va începe cu o singură unitate (*Necromancer*). Acesta va folosi magii pentru a crea diferitele clădiri pe care jucătorul dorește să le construiască. Totuși, aceste clădiri nu pot produce singure unități de luptă, pentru aceasta fiind nevoie de corpuri neînsuflețite ale adversarilor sau ale caracterelor neutre.

În majoritatea jocurilor actuale, moartea unei unități de tip *undead* înseamnă eliberarea sufletului. Totuși, pentru a permite și lupte *undead vs. undead*, sufletele unor unități de acest tip nu vor fi eliberate, iar cadavrele vor putea fi folosite pentru a crea noi unități.

Unitățile

În *Warcraft III* numărul de unități va fi mult mai mic decât în cazul celorlalte jocuri de strategie produse de *Blizzard*. Numărul de unități va diferi în funcție de rasă, vor fi mai puține unități, dar mai puternice.

Mai multe unități și eroi din *Warcraft II* își vor face apariția și în noul joc, dar vor fi o mulțime de unități noi.

La fel ca și în *Starcraft*, se vor putea obține *upgrade*-uri și magii care se vor aplica pentru toate unitățile de un anumit tip.

Ca și în versiunile anterioare, vor apărea diferite unități neutre, care se vor plimba pe hartă. Până acum acești monștri erau inofensivi, dar în *Warcraft III* unii vor fi agresivi, la fel ca și cei din *Age of Empires*.

În acest joc, ca și în *Warcraft II*, vor fi și unități acvatice și aeriene. S-ar putea să existe și o unitate aeriană care va trebui să aterizeze pentru a putea ataca.

Vor reapărea și dragonii, dar ei vor fi complet diferiți față de cei din *Warcraft II*.

O nouă categorie de unități este formată din giganți. Aceștia vor fi afectați în mod diferit de anumite magii sau alte efecte.

Unitățile obișnuite nu vor putea fi echipate cu arme, armuri sau diferite alte obiecte. Chiar dacă acestea vor apărea în joc, doar eroii le vor putea folosi.

Eroii

Eroii sunt unități speciale cu diferite caracteristici. Ei au posibilitatea de a avansa în niveluri, colectează obiecte și oferă beneficii speciale celorlalte unități aflate în apropiere.

Fiecare erou va avea caracteristici proprii printre care abilități și magii unice. În timpul jocului, eroii se vor dezvolta pe măsură ce vor câștiga bătălii sau vor găsi obiecte noi. Pe măsură ce nivelul eroilor va crește, punctele de viață și capacitatea de atac vor crește. De asemenea, pentru fiecare nivel, jucătorul va putea alege o nouă abilitate pentru erou.

Atributele eroilor vor fi puterea (*Strength*), agilitatea (*Agility*) și inteligența (*Intelligence*). Fiecare dintre acestea va influența viteza de mișcare a eroului, capacitățile de luptă și posibilitățile de folosire a magiilor. Valorile acestor atribute vor crește pe măsură ce eroul va câștiga bătălii și astfel acesta va deveni mai puternic.

În funcție de tipul eroului (*Warlord*, *Paladin* etc) valorile atributelor vor crește automat. Jucătorul nu va putea alege care valori vor crește. De exemplu, pentru eroii care au abilități de magie dezvoltate, va crește, în principal, inteligența, în timp ce pentru cei care au capacități de luptă, va crește puterea. Pe măsură ce nivelul eroilor va crește, diferențele dintre clasele de eroi se vor face simțite.

În *Warcraft III* eroii sunt nemuritori. Dacă un erou este învins în luptă, el va putea fi recrutat din nou.

După recrutare eroul va avea același nivel ca și înainte de începerea luptei și aceleași posibilități de a folosi magiile.

Clădirile

În *Warcraft III* vor exista două tipuri de clădiri. Pe de o parte sunt clădirile pe care le construiește și le controlează jucătorii și care se comportă la fel ca în *Starcraft*. Pe de altă parte există clădiri neutre care există pe hartă de la începutul jocului și cu care nu pot interacționa decât eroii.

Se pare că dimensiunile clădirilor din *Warcraft III* vor fi puțin mai mari decât ale celor din *Starcraft*.

Vor exista diferite tipuri de clădiri cum ar fi cele prin intermediul cărora sunt recrutate unități noi sau cele folosite pentru apărare (ziduri, turnuri etc.)

Resursele

În *Warcraft III* aurul va fi principala resursă folosită pentru a recruta unități sau a construi clădiri. Probabil că lemnul și petrolul din *Warcraft II* nu vor dispărea și vor apărea și alte tipuri de resurse.

Jocul multiplayer

Deocamdată *Blizzard* promite că numărul de participanți la un joc în rețea va fi mai mare decât opt.

Una dintre cele mai atrăgătoare calități a jocurilor *Blizzard* era posibilitatea ca durata unui joc multiplayer să fie relativ mică. În *Warcraft III* durata medie a unui joc multiplayer va fi aceeași ca și în *Warcraft II*.

Teoretic, datorită recompenselor care sunt primite după îndeplinirea unei misiuni s-ar putea ca balanța să fie înclinată decisiv de partea unuia dintre jucători. *Blizzard* promite că va încerca să păstreze echilibrul dintre jucători și în acest caz, deci îndeplinirea unei misiuni nu va fi decisivă pentru a câștiga jocul. Totuși, îndeplinirea unei misiuni va aduce unele avantaje.

Aliații vor putea face schimburi de resurse pentru a se ajuta în timpul jocului. Ni se promit și alte posibilități de

a ne ajuta coechipierii cum ar fi schimbul de unități sau posibilitatea de a controla unitățile unui aliat.

Pentru a-și satisface fanii, *Blizzard* a promis că va adăuga pe site-ul propriu un forum pe tema jocului *Warcraft III*.

În cazul jocurilor anterioare unii jucători se foloseau de diverse metode pentru a trișa. *Blizzard* promite să rezolve această problemă pentru ca jocurile să fie corecte.

Editorul de hărți

Jucătorii vor avea posibilitatea de a-și crea propriile hărți folosind un editor pus la dispoziție de *Blizzard*.

Noul editor este mult mai avansat decât cel din *Starcraft*. Pentru jucătorii care posedă unele cunoștințe de programare există un limbaj de *scripting* care permite un mai bun control al evenimentelor care apar pe parcurs. Evident, editorul va putea fi folosit și de cei care nu posedă astfel de cunoștințe.

Editorul permite chiar și adăugarea de filme, fiecare jucător putând crea astfel o poveste *Warcraft* proprie.

Mai multe despre unități

Blizzard a dat publicității mai multe detalii "istorice" despre diferitele unități care vor apărea în *Warcraft III*. În cele ce urmează vă vom prezenta câteva dintre aceste descrieri.

Dwarven Rifflemen

Aceștia sunt renumiți de mult timp pentru modul în care prelucrează oțelul, dar și pentru capacitățile de luptă. Cu ajutorul inventivilor gnomi ei au construit arme noi și mortale; celebrul *Dwarven Demolition Squad* a găsit modalități noi de folosire a explozibilului din dotare, efectele acestuia fiind mult mai diverse.

Human Paladins

Nobilii războinici ai umanității posedă atât calități magice deosebite, cât și o mare putere de luptă. Ei folosesc armele și armurile strămoșilor lor și cutreieră din oraș în oraș pentru a corecta greșelile societății în care trăiesc. Chiar dacă *Alianța* se află în pragul distrugerii, acești războinici au pus din nou mâna pe arme pentru a lupta cu inamicii umanității.

Orc Wolf Riders

Orcii au renunțat la magiile demonice și acum practică vechea magie șamanistică a nobililor lor strămoși. În timp ce șamanii orci sunt capabili să folosească furia naturii pentru a-și lovi inamicii, ordinul *Wolf Riders* a refăcut legătura ancestrală cu monștrii zăpezii și sunt din nou pregătiți să participe la lupte, instalând frica în inimile dușmanilor.

Minotaurs

Învăluiți de mister, minotaurii din *Barrens* își distrug inamicii, strivindu-i sub copite. Legenda spune că pentru fiecare dușman ucis în luptă, un războinic minotaur își împletește o șuviță de păr de pe coama sa aspră, ca semn al curajului și puterii sale deosebite. Puțini dintre cei care s-au confruntat cu un minotaur nu au determinat apariția unei astfel de șuvițe.

Human Sorceresses

Aceste vrăjitoare au venit din *Dalaran* pentru a avea grijă de *Battle Magi*. Ele își folosesc puterile magice ori de câte ori umanitatea se află în pericol. Cu toate că abilitățile lor sunt folosite foarte rar direct în luptă, ele pot ajuta războinicii *Alianței* cu o varietate mare de vrăji vindecătoare sau care provoacă iluzii inamicului.

Orc Grunts

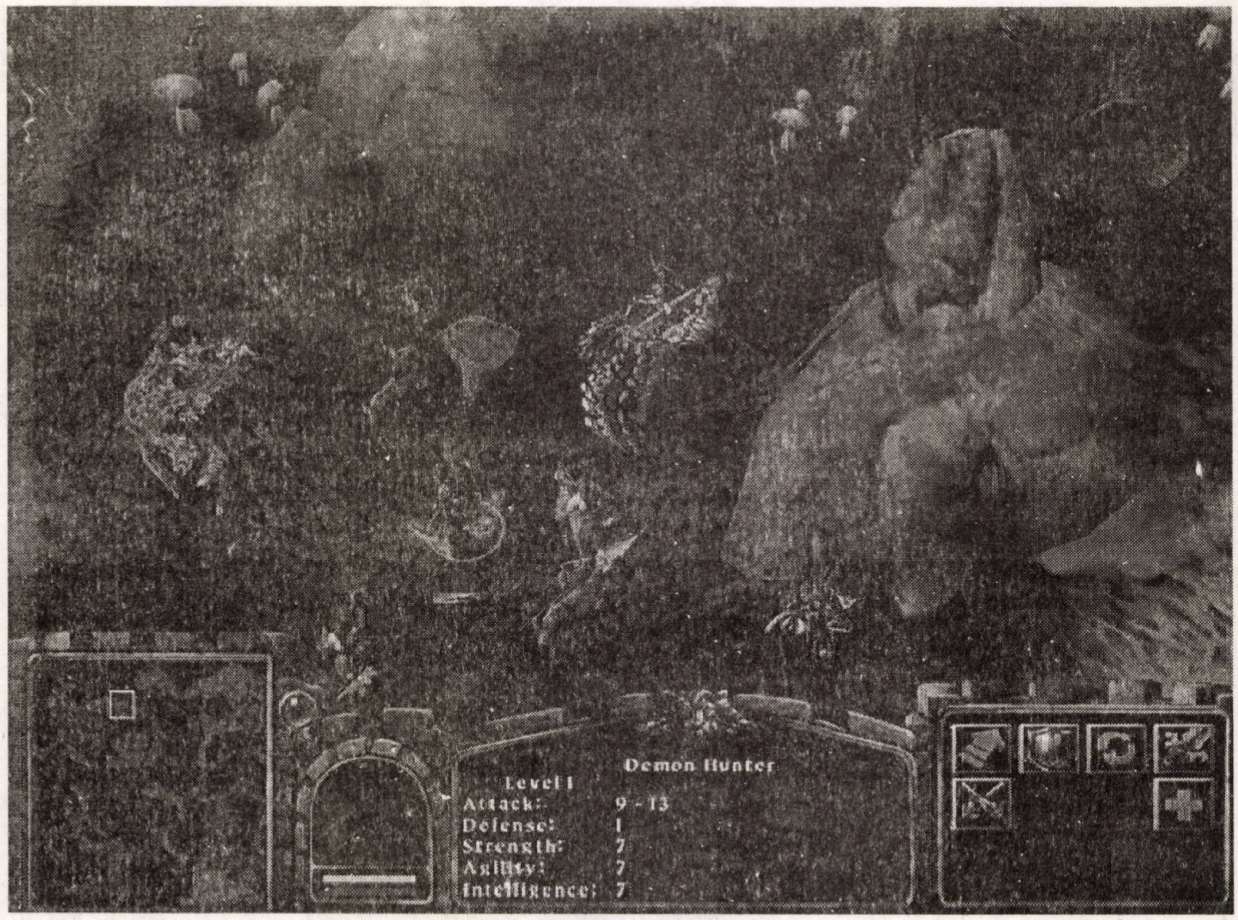
Ei reprezintă prima și ultima linie de apărare a *Hoardei*. Acești luptători puternici folosesc topoare de luptă și arată toată sălbăticia și cruzimea rasei lor. De-a lungul ultimelor generații cele mai importante caracteristici ale lor au fost depravarea și brutalitatea. Acum însă, sub conducerea vizionară a lui *Thrall*, ei s-au apropiat de imaginea nobilă a strămoșilor.

Wyverns

Aceste zburătoare au fost bucuroase să se alăture noii *Hoarde* șamanistice. este posibil ca ele să fi fost impresionate de orci sau doar să fie fidele angajamentului lor pentru onoare. Cert este că ele au fost de acord să fie folosite de orci pentru a lupta împotriva celor care le deranjează pacea ținutului lor. Ele au aceeași strămoși cu dragonii și grifonii, și își utilizează moștenirea genetică împotriva inamicilor din aer sau de pe sol.

Death Knights

Acești călăreți au fost creați de către *Gul'dan* în timpul celui de-al doilea război. El a folosit cadavrele cavalerilor umani și spiritele malefice eliberate din trupurile membrilor ordinului *Orc Warlocks*. A apărut astfel un nou ordin care l-a înlocuit pe cel vechi. Cavalerii resuscitați au devenit principalii utilizatori ai magiei din *Hoardă*. Totuși, nesupunerea și apatia cu care luptau pentru cauza orcilor i-a făcut nedemni de încredere. Ei și liderul lor *Ner'zhul* au fost capturați de către *Legiune* în timpul războiului din *Draenor*. Acum ei apar din nou în luptă și se numără printre cele mai puternice unități din universul *Warcraft III*.



Concluzie

În scurt timp universul *Warcraft* se va îmbogăți cu un nou joc care promite să se ridice mult deasupra nivelului predecesorilor săi.

Se pare că acest joc merită calificativul de cel mai așteptat joc al anului 2001, deoarece mulți speră că ziua în care vor putea să pornească la luptă de partea uneia dintre cele patru rase este apoape.

Probabil că mulți își vor petrece o mare parte a vacanței de vară în fața monitorului, încercând să descopere noi strategii prin care să își învingă adversarii din întreaga lume. Noi nu putem decât să vă avertizăm că petrecerea unui interval mare de timp în fața ecranului are efecte nedorite. Știm însă că nu vă pasă, așa că nu vom mai insista.

Dacă doriți să aflați mai multe amănunte referitoare la acest joc vă recomandăm să vizitați *site*-ul oficial al companiei *Blizzard* (**www.blizzard.com**). Aici puteți găsi informații actuale referitoare la progresul relizat de producători, precum și diferite alte știri deosebit de interesante.

Cei de la *Blizzard* au promis că jocul nu va necesita resurse hardware de ultima generație, chiar și cei cu calculatoare mai vechi putându-se bucura de el.

Totuși, este puțin probabil ca jocul să ruleze în condiții acceptabile dacă nu dispuneți de un procesor *Duron*, *Athlon*, *Celeron Pentium III* sau *Pentium 4*.

De asemenea, este nevoie de o placă video performantă pentru că, deși calitatea graficii este deosebită, nu o veți putea admira dacă placa video de care dispuneți nu va fi capabilă să o redea.

Nu trebuie neglijate nici efectele sonore, o placă audio performantă fiind recomandabilă pentru acest joc.

Nu în ultimul rând, farmecul acestui joc este mult redus dacă nu vă veți putea confrunta cu prietenii. Dacă nu sunteți conectat într-o rețea locală, singura posibilitate de a vă demonstra abilitatea rămâne *Internet*-ul. Pentru acest lucru aveți nevoie de un simplu modem și de un furnizor de servicii *Internet*.

În final

La un moment dat, în revista *PC Games*, se spunea că expresia "*la sfântu'-așteaptă*" ar putea fi înlocuită cu "*la lansarea lui Warcraft III*". Noi nu putem decât să sperăm că de această dată lansarea nu va mai fi amânată și la vară vom putea porni în noua aventură *Warcraft*.

Așadar, înarmați-vă cu multă răbdare și așteptați lansarea oficială a jocului. Vă urăm, încă de pe acum, mult succes în distrugerea raselor inamice! Dar, nu uitați nici o clipă că aceste jocuri vă poartă în lumi virtuale, iar voi trebuie să vă confruntați cu grijile voastre reale, mult diferite de cele din jocuri.

Carmen Șușman este redactor al GInfo. Poate fi contactată prin intermediul redacției.

NETREPORT

PCREPORT ÎN MILENIUL III

www.pcreport.ro

Conectat la viitorul digital

COMPUTER PRESS AGORA Str. Tudor Vladimirescu nr. 63/1, Tg. Mures, CP 230/11, 4300

Tel: 065-166516, FAX: 065-166290, e-mail: office@agora.ro

PC MAGAZINE ROMÂNIA

COMPUTER PRESS AGORA

Din acest număr:

- **Vedete in 2000**
- **Internet business**
- **Premiile de excelenţă tehnică**
- **Laborator**

www.pcmagazine.ro

Revista utilizatorilor de calculatoare personale

COMPUTER PRESS AGORA

Str. Tudor Vladimirescu nr. 63/1, Tg.Mureş (CP 230/1), 4300
Tel. 065-166516, Fax 065-166290, e-mail: office@agora.ro

Sucursala Bucureşti: Str. Poterași 35/2 sec.IV (CP 94/49) 75144 Bucureşti
Tel.: 01-3309282, Fax: 01-3309285

gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

Martie 2001
Vol. 11/3

Lei 15.000

din sumarul numărului:

FOCUS

ARBORI BINARI
de formă arbitrară

TEHNO

SYSmark 2000

BEGIN

AVOCAȚII

Microsoft®

BABEL sisteme nestandard

BEGIN conjectura palindroamelor

Datorită noului Armada M700 cu procesor Intel® Pentium® III 750 MHz poți acum să faci față oricăror situații neprevăzute. Poți fi în mai multe locuri în același timp și poți face mai multe lucruri deodată. Pentru că Armada M700 are o autonomie de funcționare de peste 5 ore prin folosirea simultană a două baterii iar tastatura Easy Access Internet îți dă posibilitatea conectării imediate la Internet. În plus, cântărește mai puțin de 2,3 kg și are o grosime de numai 3 cm. Astfel că treaba pe care ai început-o la birou o poți termina după program, la cinema. Tehnologia care te inspiră. De la Compaq. Bun venit în noua eră IT.

COMPAQ
Inspiration Technology

EȘTI LA BIROU.
ȘI TOTUȘI NU EȘTI...

COMPAQ ARMADA M700

Procesor Intel® Pentium® III 750MHz • 128MB RAM
• hard disk 12GB • DVD • modem 56k • display TFT 14"
• Ethernet 10/100 • Microsoft Windows 2000/NT4

intel inside
pentium® III

scop®
COMPUTERS

Barbu Văcărescu 162
Tel: 231 65 78
Fax: 231 65 96
www.scop.ro

© 2001 Compaq Computer Corporation. COMPAQ and the Compaq logo Registered in U.S. Patent and Trademark Office. Armada and Inspiration Technology are trademarks of Compaq Information Technologies Group, LP. in the U.S. and other countries. Intel, the Intel Inside Logo and Pentium are registered trademarks of Intel Corporation.

Vol. 11, Nr. 3 /2001

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Redactor:
Carmen Șușman (Alba Iulia)

Au mai colaborat:
Laviniu Bejenaru (Dr. Tr. Severin)
Bazil Pârv (Cluj)
Tiberiu Socaciu (Cluj)
Adrian Vasilescu (Canada)

Tehnoredactare:
Călin Borz

Coperta și grafica:
Adrian Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca

Adresa e-mail: office@ginfo.ro
Tel: 064/192.422
065/166.516
Fax: 065/166.290
http://www.ginfo.ro

ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359

Editor: Computer Press Agora SRL
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihaela Szabo (mszabo@agora.ro)
Ioana Udroiu (iudroiu@agora.ro)

Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

sumar

probleme.. PAG. 5

propuse pentru REZOLVARE

Olimpiada de informatică ~faza locală - Cluj~

babel.. PAG. 33

SISTEME nestandard. METODE noi de rezolvare

Laviniu BEJENARU

concurs.. PAG. 9

Rezultatele rundei a patra

Clasament

Runda a șasea

begin.. PAG. 39

Conjectura PALINDROAMELOR

Tiberiu SOCACIU

focus.. PAG. 19

Arbori BINARI de formă ARBITRARĂ

Adrian VASILESCU

Structuri de date

tehno.. PAG. 41

Performanțele PROCESOARELOR

Mihai SCORȚARU

SYSmark 2000

serial.. PAG. 29

Structură și stil în PROGRAMARE

Bazil PÂRV

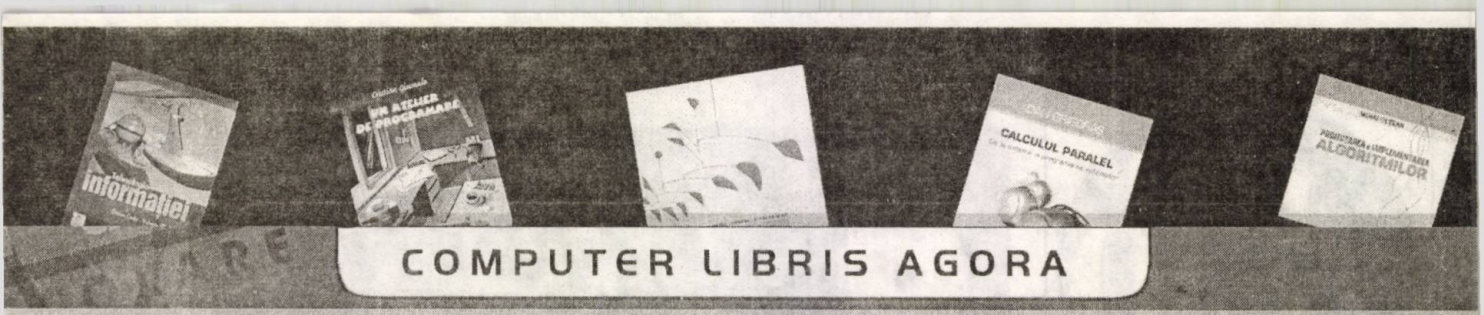
Ultimul episod

enjoy.. PAG. 47

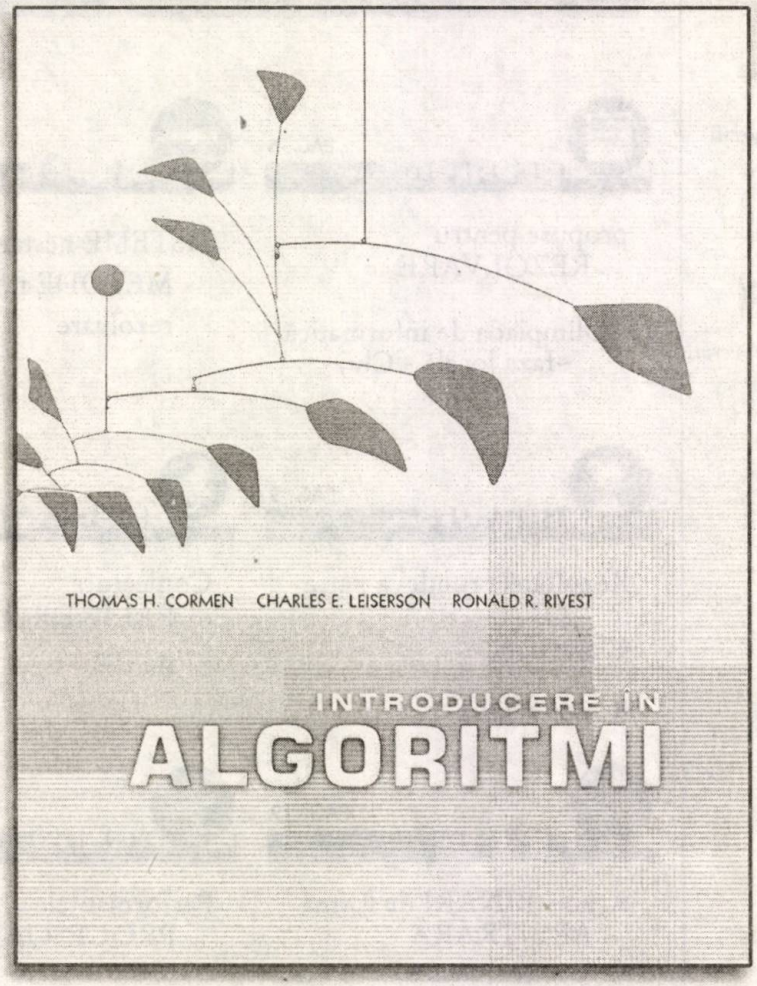
Avocații MICROSOFT

Carmen ȘUȘMAN

O întâmplare imaginară petrecută la bordul navei stelare USS Enterprise



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Micul PRINȚ

Poate v-a surprins faptul că, în numărul anterior, printre colaboratorii revistei noastre a apărut un personaj misterios: *Micul Prinț*...

Cine o mai fi și acesta? Cine se ascunde sub acest pseudonim? Redactorul șef sigur nu. Să fie redactorul șef adjunct? Nu e nici mic, nici prinț...

Micul Prinț este, după cum bine știți, eroul unei povestiri sublime. *Antoine de Saint Exupéry* l-a creat sub imboldul unei inspirații poetice, dar pesimiste.

Nu sunt filolog, nu mă voi aventura în analiza unei opere literare, dar nu pot să vă las total în ceață; trebuie să vă spun de ce anume ne-am amintit de "*Micul Prinț*".

Trăim într-o lume pe care *Saint Exupéry* nu a cunoscut-o dar a intuit-o și a prevestit-o și al cărei dramatism îl trăim zilnic din plin. Indiferența și superficialitatea ne izbesc, ne chinuie și ne amenință.

Dezamăgirile provocate de încercarea unora (și de multe ori, de reușita lor) de a ne păcăli, ne umplu sufletele de revoltă. Și totuși, fiecare om visează, dacă nu în somn - treaz, cu siguranță. Și presupun că toți visează "o lume mai bună".

Îmi este imposibil să cred că cineva visează (își dorește) o lume în care natura este sufocată și distrusă, în care oamenii sunt răi, nu doar pe ascuns (...) dar și pe față și fiecare dorește răul celuilalt (un fel de "*Familia Addams*"). Dacă îți dorești să fii fericit păcălind, oare nu îți este frică să fii păcălit la rândul tău?

Da, sunt convinsă că toți visează o lume mai bună. Dar este oare suficient să visăm? Mă simt obligată să mă opresc. Da, se pune întrebarea: *ce înseamnă o lume mai bună sau dacă este vorba de o anume persoană, ce înseamnă pentru aceasta o "viață mai frumoasă"?* Știu, știu, nu m-ați ierta dacă aș continua în mod banal cu "Banii n-aduc fericirea".

Cu toții suntem de acord că pentru o viață frumoasă este nevoie de bunăstare, este nevoie de bani cu care să ne putem asigura un confort, dacă nu luxos, măcar decent. Dar nu cu "orice preț".

Cei care visează la succese, bani și glorie, sperând că le pot obține prin mijloace speculative, prin însușirea realizărilor altora, vor constata cu amărăciune că ceea ce au obținut astfel își pierde valoarea.

Secretul reușitei se ascunde în entuziasmul cu care facem ceea ce facem - entuziasm pentru însăși lupta de a ne căuta rostul, zilnic, de a ne măsura cu... absolutul.

Dar ce este idealul? Că absolutul în mod sigur este departe... Idealul, prin esența lui, este ceea ce am visat, ceea ce vrem să atingem, dar care, în mod firesc, "aleargă" în fața noastră, oricât ne-am strădui să-l "prindem". Suntem sortiți să rămânem mereu în urma "idealului", dar măcar știm în ce direcție este...

Nu vă strâmbați, nu este o oră de dirigenție. Mă preocupă să găsesc răspunsul la o întrebare jenantă. Înainte să pun întrebarea, voi face ceva ce mă face să mă simt neplăcut, deoarece am senzația că voi desconspira pe cineva, mai exact pe mai mulți "cineva".

Dragi cititori, participanți sau nu la ***Concursul de Programare Agora***, știați că mai mulți participanți la acest concurs discută pe o listă de dialog despre probleme, oferă rezolvări la cheie, sau dimpotrivă, caută rezolvări?

Unii, mai "serioși", se mulțumesc cu ideile, alții... Îmi pare rău că stilul meu categoric, poate chiar insinuant, sugerează deja, înainte să pun întrebarea cu pricina, că nu mă bucur de această activitate în echipă.

Dar să pun întrebarea: *este sau nu este corect să participi la un concurs individual de programare, dar rezolvările tale să fie puse la punct public de către mai mulți, eventual de unul singur, dar de un altul, deci nu de tine, care trimiți rezolvarea și o semnezi?*

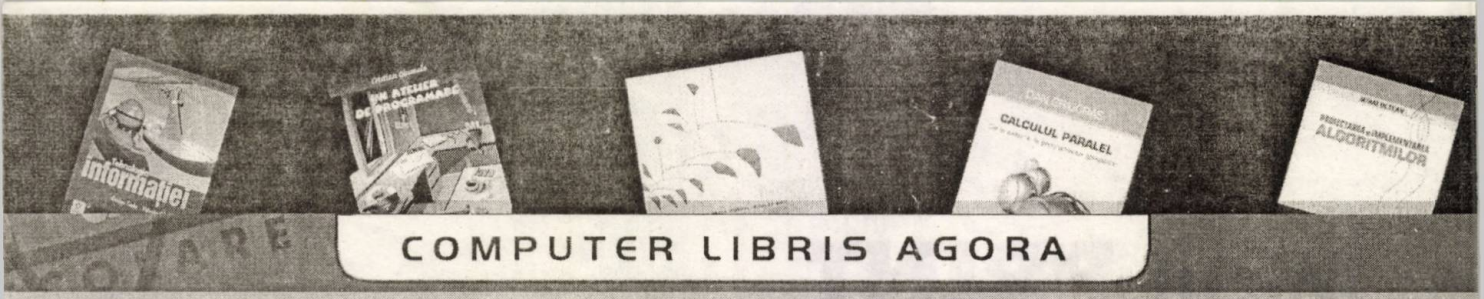
Îl văd pe *Micul Prinț* care își clatină capul blond, fularul îi flutură dezaprobator. Bunul simț al nostru, al fiecăruia, ne spune, de asemenea, că nu este corect.

Degeaba vă ascundeți după paravanul beneficiilor Rețelei. Știu că argumentele pro încearcă să convingă că "trebuie să fii prost să nu comunici via *Internet*" dacă tot există această posibilitate. Și? Ce facem cu cei care nu au auzit de această listă? Ghinionul lor?

Noi, organizatorii concursului, am pornit la drum cu convingerea că acest concurs este unul individual (de altfel chiar și *ACM*-ul, un concurs pe echipe se desfășoară cu doar 3 (trei) membri într-o echipă.

Ne vedem, oricum, după cele șase runde, la *Cluj* (la concursul în timp real), și dacă se dovedește că unii (beneficiari ai listei) nu reușesc să țină pasul, cât de cât, cu adevărații concurenți individuali, anul viitor nu ne vom mai obosi cu acest concurs. Poate inventăm unul în echipă... Dar nu putem continua, prefăcându-ne că totul decurge în mod elegant și că nu s-a întâmplat nimic.

Clara Ionescu



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.

Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

AGORA

Încercați-vă puterile!

PROBLEME propuse pentru REZOLVARE

Vă prezentăm enunțurile problemelor de la faza locală a Olimpiadei de informatică din județul Cluj. Au fost în total șase probleme, două pentru clasa a IX-a, două pentru a X-a și două pentru clasele a XI-a și a XII-a.

P030101: Tăierea cozii

Prin *tăierea cozii* se înțelege operația de transformare aplicată unui șir de caractere care constă în eliminarea ultimelor k elemente ale șirului (cu k mai mic sau egal cu jumătatea lungimii șirului). Tăierea cozii se poate face numai dacă primele k elemente sunt identice cu ultimele k, citite de la sfârșit la început.

Se citește de la tastatură un șir de lungime n ($1 < n < 256$), cu elemente din mulțimea `{0,1}`. Șirului i se pot aplica mai multe transformări succesive de tăiere a cozii.

Să se determine configurația finală minimă la care se poate ajunge după transformările posibile aplicate șirului, pentru k variind de la 1 la jumătatea lui n.

Exemple

Dacă șirul inițial este `1101011` (deci k ia valori între `1` și `3`), se obțin următoarele configurații minime:

- pentru k = 1 se obține `11010` prin două tăieri succesive.
- pentru k = 2 se obține `11010` printr-o singură tăiere.
- pentru k = 3 se obține `1101` printr-o singură tăiere.

Deci, în acest caz, configurația minimă obținută este `1101`.

Dacă șirul inițial este `1111` (deci k ia valorile `1` sau `2`), se obțin următoarele configurații minime:

- pentru k = 1 se obține `1`.
- pentru k = 2 se obține `11`.

Deci configurația minimă este `1`.

Dacă șirul inițial este `11011011` (deci k ia valori între `1` și `4`), se obțin următoarele configurații minime:

- pentru k = 1 se obține `110110` prin două tăieri succesive.
- pentru k = 2 se obține `110110` printr-o singură tăiere.
- pentru k = 3 se obține `11011` printr-o singură tăiere.
- pentru k = 4 se obține `1101` printr-o singură tăiere.

Deci, în acest caz, configurația minimă obținută este `1101`.

Notă

Citirea și tipărirea datelor se vor efectua exact ca în modelul de mai jos:

```
Date de intrare:
1101011
Date de iesire:
1101
```

P030102: Numere prime

Să se scrie un program care citește un număr natural n ($0 < n < 1.000.000.000$) corect introdus de la tastatură și tipărește toate numerele prime care se pot forma din secvențe de cifre alăturate, în ordinea apariției în număr, ale numărului dat. Într-o secvență poate intra orice număr de cifre, de la o cifră la toate. Fiecare număr va fi tipărit o singură dată, chiar dacă el se repetă ca secvență a numărului inițial.

Observație

`0` și `1` nu sunt considerate a fi numere prime.

Notă

Citirea și tipărirea datelor se vor efectua exact ca în modelele de mai jos:

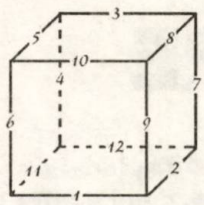
```
Date de intrare:
N=233
Date de iesire:
2 3 23 233

Date de intrare:
N=480
Date de iesire:
Nu exista solutii

Date de intrare:
N=2070
Date de iesire:
2 7
```

P030103: Cubul numerotat

Pe muchiile unui cub trebuie "plasate" numerele 1, 2, 3, ..., 12, astfel încât suma numerelor de pe cele patru muchii ale oricăror fețe să fie constantă. De asemenea, muchiile trebuie colorate cu un număr minim de culori astfel încât oricare două muchii incidente (care au un punct comun) să nu aibă aceeași culoare.

Exemplu

O soluție posibilă este



Rezultatele se vor afișa într-un fișier text **DATE.OUT** care va conține soluțiile numerotate astfel:

- pe prima linie numărul minim de culori folosite;
- pe următoarele linii numerele și culorile asociate muchiilor fiecărei fețe, sub forma (*număr - culoare*);
- fețele sunt numerotate așa cum se poate vedea mai jos.

Pentru exemplul considerat, o soluție va avea forma:

DATE.OUT

```
Nr. culori = 3
Sol. nr. 1
Fata 1.:(1 - 1) (6 - 2) (10 - 1) (9 - 3)
Fata 2.:(2 - 2) (7 - 3) (8 - 1) (9 - 3)
Fata 3.:(3 - 1) (4 - 2) (12 - 1) (7 - 3)
Fata 4.:(4 - 2) (5 - 3) (6 - 1) (11 - 3)
Fata 5.:(3 - 1) (5 - 3) (10 - 1) (8 - 2)
Fata 6.:(1 - 1) (2 - 2) (12 - 1) (11 - 3)
```

P030104: Eliminare palindroame

Se dau n numere naturale formate din oricâte cifre; acestea sunt conținute în fișierul text **DATE.IN**. Se cere curățarea șirului de elementele palindroame. Mai întâi se elimină numerele palindroame; apoi se concatenează două câte două numerele rămase. Se repetă procedeul (eliminare și concatenare) până când nu mai există numere palindroame.

Exemplu

Conținutul fișierului **DATE.IN** este următorul:

```
1 15 51 1 7 23 32 14 29 17
```

După eliminarea numerelor palindroame șirul devine `15 51 23 32 14 29 17`. După concatenare șirul este `1551 2332 1429 17`. După a doua eliminare șirul devine `1429 17`, iar după concatenare `142917`.

Fișierul de ieșire **DATA.OUT** va conține configurația finală a șirului. În cazul nostru, în final, șirul va conține un singur număr și anume `142917`.

P030105: Băcănie

Băcanul din orașul dumneavoastră încearcă o nouă metodă de afișare a prețurilor produselor pe care le are în stoc. În loc de a marca, pentru fiecare articol, prețul lui (în lei), etichetele băcanului au prețuri comparative față de alte produse. De exemplu, untul poate fi etichetat cu `unt=margarina+100`, iar margarina cu `margarina=cafea-111`. Unele articole sunt marcate chiar cu prețul corespunzător.

Datele de intrare se află în fișierul **BACANIE.IN** și sunt organizate astfel:

- toate articolele au denumiri formate din cel mult 10 litere;
- există cel puțin unul și cel mult 100 de articole;
- liniile de intrare indică fie direct prețul unui articol, fie indică faptul că acest preț este egal cu prețul unui alt produs `+/-` o sumă (în lei);
- toate prețurile care apar în fișier sunt numere naturale mai mici decât `1000`;
- toate liniile sunt corecte din punct de vedere sintactic;
- fiecare articol apare o singură dată în stânga semnului `'='`;
- articolele se află câte unul pe o linie din fișier;
- în fișierul de intrare nu există spații.

Fișierul de ieșire **BACANIE.OUT** va conține, pentru fiecare articol, câte o linie de forma `articol=pre` . Dacă prețul unui produs nu poate fi dedus pe baza datelor din fișierul de intrare, linia corespunzătoare va avea forma `articol=blank`. Ordinea articolelor din fișierul de ieșire nu are importanță.

Exemplu

BACANIE.IN

```
lapte=zahar-125
faina=225
zahar=faina+10
cafea=ceai
```

BACANIE.OUT

```
cafea=blank
faina=225
lapte=110
zahar=235
ceai=blank
```

P030106: Robot

Se dă fișierul de intrare **ROBOT.IN** (corect construit) în care:

- pe prima linie se află două numere naturale m și n ($0 < m$, $n < 10$).
- pe fiecare dintre următoarele m linii se află câte un șir de n numere din mulțimea `{0,1}`, despărțite prin câte un spațiu.
- pe linia următoare se află trei valori x, y, z, despărțite prin câte un spațiu ($0 < x \le m$, $0 < y \le n$, $z \in$ `{'N','S','E','V'}`).

Interpretarea valorilor este următoarea:

- m: numărul de linii ale unei matrice care codifică un labirint;
- n: numărul de coloane al matricei care codifică labirintul;
- valoare `1`: culoar (robotul poate ocupa această poziție);

- valoare 0: zid (robotul nu poate ocupa această poziție);
- x: linia pe care se află inițial robotul;
- y: coloane pe care se află inițial robotul;
- z: - valoarea 'N': inițial robotul privește spre prima linie a matricei;
  - valoarea 'S': inițial robotul privește spre ultima linie a matricei;
  - valoarea 'E': inițial robotul privește spre ultima coloană a matricei;
  - valoarea 'S': inițial robotul privește spre prima coloană a matricei.

  În plus, (x, y) este o poziție în care se află un culoar.

Robotul are la dispoziție doar următoarele mișcări:

- MOVE - pentru deplasarea cu o poziție pe sensul de mers (mutarea este posibilă dacă în poziția în care se ajunge nu este zid);
- ROTATE - pentru rotirea cu 90° în direcția acelor de ceasornic (robotul rămâne pe aceeași poziție dar își schimbă direcția de mers; mutarea este întotdeauna posibilă).

Programele trebuie să tipărească în fișierul **ROBOT.OUT** cea mai scurtă succesiune de mișcări pe care trebuie să le execute robotul pentru a ieși din labirint (pentru a ajunge pe prima sau ultima linie, sau pe prima sau ultima coloană). Dacă există mai multe secvențe egale, se va tipări una singură. Fiecare mișcare va fi scrisă pe câte o linie. Ultima linie va conține mesajul `Robotul a iesit din labirint in k miscari`, unde k reprezintă numărul mișcărilor. Dacă robotul nu poate ieși din labirint, în fișierul de ieșire se va scrie doar mesajul `Nu exista solutie`.

Exemple

Să considerăm următorul fișier de intrare:

ROBOT.IN

```
4 4
0 0 0 0
1 1 1 1
1 1 1 0
3 3 N
```

O soluție corectă este următoarea:

ROBOT.OUT

```
MOVE
ROTATE
MOVE
Robotul a iesit din labirint in 3 miscari
```

O altă soluție corectă este:

ROBOT.OUT

```
ROTATE
ROTATE
MOVE
Robotul a iesit din labirint in 3 miscari
```

Să considerăm acum următorul fișier:

ROBOT.IN

```
1 1
1
1 1 W
```

În acest caz singura soluție corectă este:

ROBOT.OUT

```
Robotul a iesit din labirint in 0 miscari
```

Iată și un exemplu pentru care nu există soluție:

ROBOT.IN

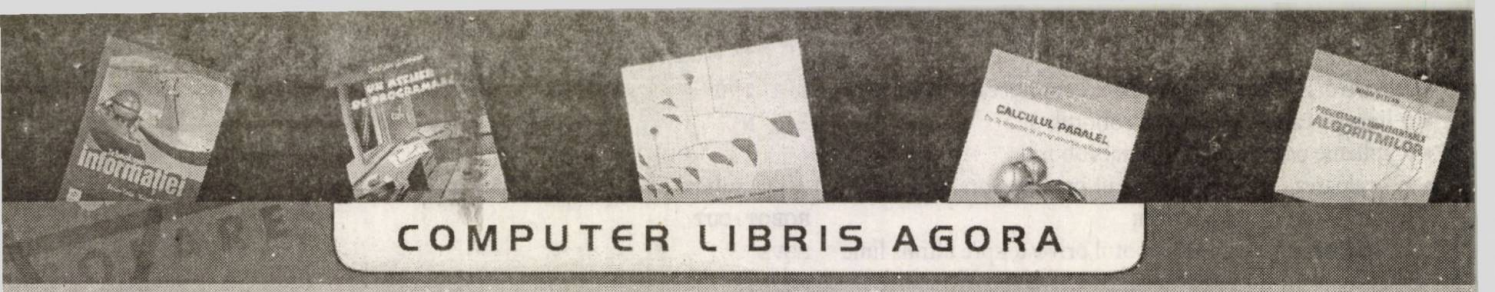
```
3 3
0 0 0
0 1 0
0 0 0
2 2 S
```

ROBOT.OUT

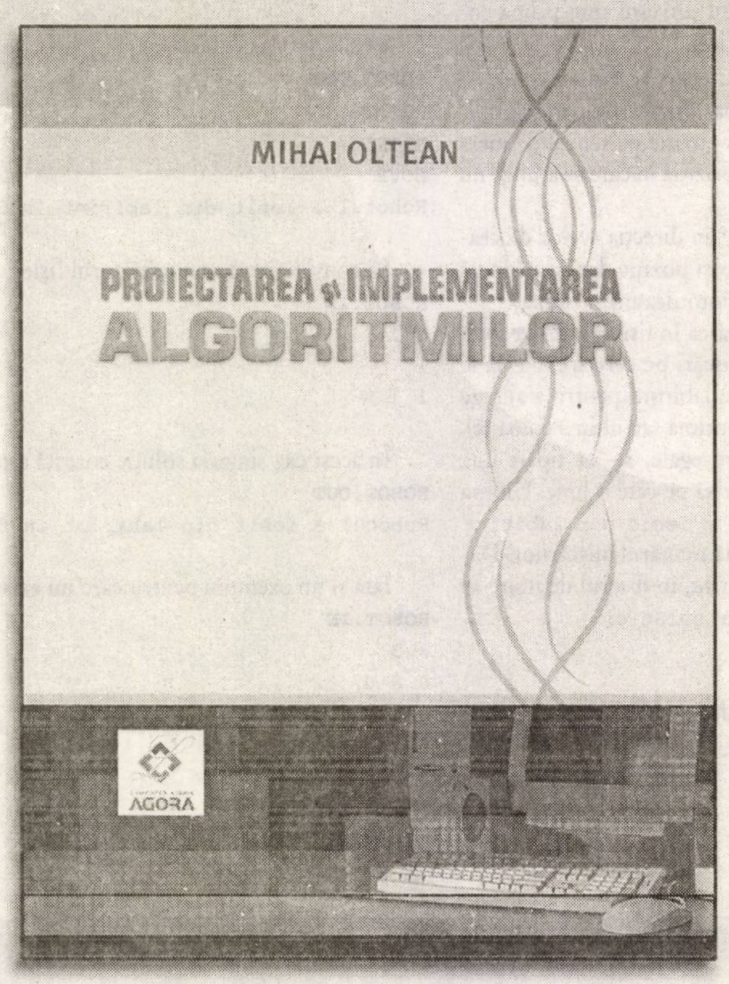
```
Nu exista solutie
```

Știați că...

- O mulțime de concurenți intră în criză de timp deoarece se prezintă la concurs fără a avea la ei un ceas.
- Multe programe nu mai funcționează corect datorită unor optimizări efectuate în ultimul moment și ale căror efecte nu au mai fost testate.
- Mulți concurenți încearcă să rezolve la început problema cea mai dificilă și apoi nu mai au timp să lucreze și la celelalte.
- Cazul cel mai defavorabil nu corespunde întotdeauna dimensiunilor maxime ale datelor de intrare.
- Multe programe funcționează corect pentru majoritatea datelor de intrare posibile, dar furnizează erori neașteptate pentru combinații particulare cum ar fi dimensiunile minime ale datelor de intrare.
- Cea mai enervantă eroare care poate apărea într-un program este scrierea incorectă a numelui fișierului de intrare sau a celui de ieșire.
- O rezolvare nu trebuie să fie neapărat corectă; ea trebuie să furnizeze rezultatul corect doar pentru seturile de date folosite de comisie la evaluare.



COMPUTER LIBRIS AGORA



"Cum aș putea să justific existența acestei cărți? Cum aș putea spre exemplu să justfic cele 100 de pagini despre programarea dinamică, acum când această metodă nu mai figurează în programele școlare? Nu știu. Sau, cum aș putea să justific capitolul despre algoritmi genetici, când pe plan mondial abia de câțiva ani a fost dovedită utilitatea și eficiența lor? Din nou nu știu! Dar rămâne bănuiala: a scrie algoritmi înseamnă a crea. Și nu poți crea fără să gândești."

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

CONCURSUL de programare AGORA

Cea de-a patra rundă a Concursului de Programare Agora s-a încheiat. În acest număr vă vom prezenta soluțiile oficiale ale celor trei probleme, precum și rezultatele obținute de concurenți la această etapă. De asemenea, veți putea vedea care este clasamentul intermediar, după patru etape.

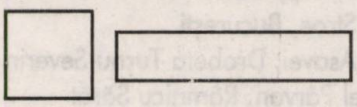
Cele trei probleme ale rundei a patra au fost relativ simple dar, surprinzător, doar șase concurenți au reușit să obțină toate cele 300 de puncte puse în joc.

Capcana

După cum se poate vedea în clasamentul etapei, există o mulțime de concurenți care au acumulat 297 puncte, adică au pierdut doar punctele corespunzătoare unui test. Este vorba despre un test de la problema *Asemănare* care i-a pus în dificultate pe mulți participanți.

Această problemă cerea verificarea asemănării a două poligoane. Generalizând unul din cazurile de asemănare pentru triunghi, un număr foarte mare de concurenți au considerat că *două poligoane sunt asemenea dacă au unghiurile respectiv congruente.*

Totuși, în realitate, această afirmație nu este adevărată, după cum se poate vedea în figura de mai jos.



Este evident că toate unghiurile au 90°, deci unghiurile sunt respectiv congruente. Totuși, este la fel de evident că cele două poligoane nu sunt asemenea.

Aceasta a fost cea mai "frumoasă" capcană în care au căzut concurenții la această rundă. Cei care nu au fost atenți la acest detaliu ar trebui să fie foarte bucuroși deoarece a fost ales un singur test de acest tip.

Liste de discuții

Existența diferitelor liste de dialog unde se discută despre *Concursul de Programare Agora* nu poate decât să ne bucure. Există însă concurenți care nu sunt de acord cu faptul că pe aceste liste apar și idei de rezolvare pentru probleme. Ei sunt nemulțumiți că se chinuie să rezolve anumite probleme, în timp ce alții așteaptă ca altcineva să le trimită o idee de rezolvare.

Noi considerăm că un concurent care nu rezolvă singur nici o problemă, nu are șanse prea mari să se claseze între primii zece. De aceea, vă asigurăm că un astfel de concurent nu va participa la etapa finală.

Totuși, principalul scop al *Concursului de Programare Agora* este acela de a vă ajuta să vă pregătiți pentru concursurile de programare "reale" la care doriți să participați. Din acest motiv, credem că listele care au ca subiect problemele de la concursul nostru, constituie un element pozitiv.

Cu toate acestea, după părearea noastră, cei care preiau o rezolvare a unui alt concurent și o trimit ca fiind rezolvare proprie nu procedează corect. În plus, există riscul ca acel concurent să fie descalificat deoarece soluția este mult prea asemănătoare cu cea a unui alt concurent. Situația este diferită în cazul în care este preluată doar ideea de rezolvare și concurentul implementează un algoritm propriu.

În concluzie, considerăm că aceste liste de discuții au un rol foarte important, dar nu ar trebui să se exagereze cu prezentarea soluțiilor.

De exemplu, nu ne-am bucurat atunci când pe una dintre listele de discuții a apărut exact formula care trebuia folosită pentru rezolvare. Din nefericire pentru alții, pe o altă listă s-a ales o formulă incorectă.

Descalificări

Anul trecut, obișnuiam să publicăm numele celor descalificați în paginile revistei. Am decis ca de data aceasta numele acestora să nu mai fie publice, concurenții descalificați fiind înștiințați prin e-mail asupra deciziei redacției. Până acum au fost descalificați 27 de concurenți.

Runda 4 - Top 50

Csaba Andras, Oradea	**300**
Ciprian Baicu, Drobeta Turnu Severin	**300**
Vencel Bors, Oradea	**300**
Cosmin-Silvestru Negruseri, Bistrița	**300**
Csaba Patcas, Oradea	**300**
Radu Ștefan, Brașov	**300**

Mugurel Ionuț Andreica, București	**297**
Aurelian Ghiță, Buzău	**297**
Mihai Hărănguș, Cluj-Napoca	**297**
Maximilian Machedon, București	**297**
Liviu Păunescu, Constanța	**297**

12. George Drumea, București	294
13. Bogdan Stan, Câmpina	293
14. Alexandru Mosoi, Bacău	292
15. Adrian Cârcu, Bistrița	290
16. Lucian-Daniel Stanciu, Brăila	286
17. Mihai-Silviu Chiru, Brăila	281
18. Artur Kuczapski, Oradea	280
19. Ciprian Cană, Vaslui	275
20. Lucian-Raul Silistru, Vaslui	272
21. Cristian Alexandrescu, Botoșani	262
22. Ștefan Bucur, București	261
23. Petru-Simon Moț, Brăila	250
24. Dan Ghinea, București	247
25. Cătălin Kesa, Bușteni	238
26. Alexandra-Elena Constantin, Ploiești	237
27. Adrian Cearnău, București	235
27. Andrei Csibi, Cluj-Napoca	235
27. Mihail Piț-Rada, Drobeta Turnu Severin	235
27. Mihai-Vlad Pantiș, Cluj-Napoca	235
31. Horea Coroiu, Cluj-Napoca	233
31. Claudiu Gruia, București	233
33. Cristian Băicoianu, Ploiești	232
34. Mihail Popa, București	228
35. Octavian-Daniel Dumitran, București	226
35. Valentin Pop, Carei	226
35. Mihai Stroe, București	226
38. Victor Asavei, Drobeta Turnu Severin	223
38. Gabriel Pârvan, Râmnicu Sărat	223
40. Bogdan Dumitru, Ploiești	221
41. Paul Marinescu, Buzău	218
42. Bogdan Nicolae, Sibiu	215
43. Andrei-Liviu Dumbrava, Iași	210
44. Daniel Crișan, Râmnicu Vâlcea	206
45. Andrei Giurgiu, București	203
46. Elvis Asaftei, Vaslui	200
46. Horia Ciurdar, Timișoara	200
46. Vlad Dascălu, Bacău	200
46. Costin Speciac, Buzău	200
50. Cristian Toth, Lugoj	199
50. Vlad Vâlceanu, Arad	199

Sfere

Această problemă a fost rezolvată corect de către 50 dintre cei care au participat la a patra rundă a *Concursului de Programare Agora*. O metodă de rezolvarea a problemelor de acest tip a fost prezentată în articolul *Geometrie 3D* publicat în numărul 11/2 (februarie 2001) al *GInfo*.

Să presupunem că am ales k sfere; vom încerca să determinăm cu cât crește numărul de zone în care este împărțit spațiul după alegerea celei de-a $(k+1)$-a sfere.

Numărul maxim de zone se obține dacă oricare două sfere se intersectează, adică nu există nici o pereche de sfere tangente, nu există nici o sferă care să se afle complet în interiorul unei alte sfere, nu există trei sfere care să aibă un cerc comun și oricare patru sfere nu au nici un punct comun.

Sfera considerată se va intersecta cu celelalte k sfere după k cercuri. Vom încerca să determinăm acum numărul maxim de zone în care aceste k cercuri pot împărți suprafața sferei.

Deoarece dimensiunile sferelor nu au nici o importanță, putem considera că noua sferă are o rază mult mai mare decât cea a celorlalte k sfere. La limită, vom considera că raza acestei sfere este infinită. Datorită faptului că, în acest caz, sfera degenerează într-un plan (un plan poate fi considerat ca fiind o sferă de rază infinită) problema în spațiu se reduce la o problemă în plan.

Enunțul acestei noi probleme este următorul: *determinați numărul maxim de zone în care poate fi împărțit planul prin intermediul a n cercuri.*

Cercurile împart planul într-un număr maxim de zone dacă oricare două cercuri se intersectează, adică nu există nici o pereche de cercuri tangente, nu există nici un cerc care să se afle complet în interiorul unui alt cerc și nu există nici un grup de trei cercuri care să aibă un punct comun.

Considerăm că am ales k cercuri și încercăm să determinăm numărul de zone suplimentare care apar dacă alegem cel de-al $(k+1)$-lea cerc. Deoarece cercul se intersectează cu fiecare dintre celelalte k cercuri în câte două puncte, rezultă că circumferința sa va fi împărțită în $2 \cdot k$ regiuni. Ca urmare, numărul de zone suplimentare care apar după considerarea unui nou cerc este egal cu dublul numărului cercurilor considerate anterior.

Așadar, relația de recurență, cu ajutorul căreia să se poată determina numărul de zone în care poate fi împărțit un plan, este $c_{k+1} = c_k + 2 \cdot k$. Deoarece un cerc împarte planul în două zone (interiorul cercului și exteriorul său), avem $c_1 = 2$.

Astfel obținem:

$$\begin{aligned} c_n &= 2+2+4+6+8+\ldots+2\cdot(n-1) \\ &= 2+2\cdot(1+2+3+\ldots+n-1) \\ &= 2+2\cdot\frac{n\cdot(n-1)}{2} \\ &= n^2-n+2. \end{aligned}$$

Așadar, numărul maxim de zone în care k cercuri pot împărți suprafața unei sfere este de $k^2 - k + 2$. Ca urmare, prin considerarea celei de-a $(k+1)$-a sfere numărul de zone este mărit cu $k^2 - k + 2$.

Așadar, relația de recurență, cu ajutorul căreia se poate determina numărul de zone în care poate fi împărțit spațiul, este $s_{k+1} = s_k + k^2 - k + 2$.

Deoarece o sferă împarte spațiul în două zone (interiorul și exteriorul ei), $c_1 = 2$.

Efectuând calculele obținem:

$$\begin{aligned} s_n &= 2+(1^2-1+2)+(2^2-2+2)+(3^2-3+3)+\ldots+[(n-1)^2-(n-1)+2] \\ &= 2+[1^2+2^2+3^2+\ldots+(n-1)^2]-[1+2+3+\ldots+(n-1)]+2\cdot(n-1) \\ &= 2+\frac{n\cdot(n-1)\cdot(2\cdot n-1)}{6}-\frac{n\cdot(n-1)}{2}+2\cdot(n-1) \\ &= \frac{12+2\cdot n^3-3\cdot n^2+n-3\cdot n^2+3\cdot n+12\cdot n-12}{6} \\ &= \frac{2\cdot n^3-6\cdot n^2+16\cdot n}{6} \\ &= \frac{n\cdot(n^2-3\cdot n+8)}{3}. \end{aligned}$$

Așadar, numărul maxim de zone în care poate fi împărțit spațiul cu ajutorul a n sfere este $\frac{n\cdot(n^2-3\cdot n+8)}{3}$.

Versiunea oficială a rezolvării acestei probleme este următoarea:

Listing SPHERE.CPP

```
#include <stdio.h>

void main(void){
  long n;
  FILE *f=fopen("SPHERE.IN","rt");
  fscanf(f,"%ld",&n);
  fclose(f);
  f=fopen("SPHERE.OUT","wt");
  fprintf(f,"%ld",n*(n*n-3*n+8)/3);
  fclose(f);
}
```

Au rezolvat corect:

Cristian Alexandrescu, Botoşani
Csaba Andras, Oradea
Mugurel Ionuţ Andreica, Bucureşti
Liviu-Cosmin Andreicuţ, Bucureşti
Elvis Asaftei, Vaslui
Victor Asavei, Drobeta Turnu Severin
Cristian Băicoianu, Ploieşti
Ciprian Baicu, Drobeta Turnu Severin
Livia Bana, Drobeta Turnu Severin
Vencel Bors, Oradea
Ştefan Bucur, Bucureşti
Ciprian Cană, Vaslui
Adrian Cârcu, Bistriţa
Adrian Cearnău, Bucureşti
Mihai-Silviu Chiru, Brăila
Alexandra-Elena Constantin, Ploieşti
Horea Coroiu, Cluj-Napoca
Andrei Csibi, Cluj-Napoca
George Drumea, Bucureşti
Andrei-Liviu Dumbrava, Iaşi
Octavian-Daniel Dumitran, Bucureşti
Bogdan Dumitru, Ploieşti
Aurel-Mihai Fulger, Constanţa
Dan Ghinea, Bucureşti
Aurelian Ghiţă, Buzău
Vincent Groenhuis, Olanda
Claudiu Gruia, Bucureşti
Mihai Hărănguş, Cluj-Napoca
Cătălin Kesa, Buşteni
Artur Kuczapski, Oradea
Maximilian Machedon, Bucureşti
Paul Marinescu, Buzău
Mihail Piţ-Rada, Drobeta Turnu Severin
Alexandru Mosoi, Bacău
Petru-Simon Moţ, Brăila
Cosmin-Silvestru Negruseri, Bistriţa
Bogdan Nicolae, Sibiu
Mihai-Vlad Pantiş, Cluj-Napoca
Gabriel Pârvan, Râmnicu Sărat
Csaba Patcas, Oradea
Liviu Păunescu, Constanţa
Valentin Pop, Carei
Mihail Popa, Bucureşti
Lucian-Raul Silistru, Vaslui
Johannes Slotta, Germania
Costin Speciac, Buzău
Bogdan Stan, Câmpina
Lucian-Daniel Stanciu, Brăila
Mihai Stroe, Bucureşti
Radu Ştefan, Braşov

Runda 4, problema 2

Asemănare

Această problemă a fost rezolvată corect doar de către 8 dintre cei care au participat la a patra rundă a *Concursului de Programare Agora*. Problema a fost propusă spre rezolvare membrilor lotului național de informatică în tabăra de pregătire și selecție de la *Cluj*, din anul 1998. O rezolvare a acestei probleme a fost publicată și în numărul 8/6 (toamna 1998) al *GInfo*.

Potrivit definiției asemănării poligoanelor, două poligoane se vor numi ***asemenea*** dacă și numai dacă au unghiurile respectiv congruente și laturile respectiv proporționale.

Știm că în cazul particular al triunghiurilor, pentru a demonstra asemănarea este suficient să arătăm că una dintre următoarele afirmații este adevărată:

- triunghiurile au unghiurile respectiv congruente;
- triunghiurile au laturile respectiv proporționale;
- triunghiurile au două laturi respectiv proporționale și unghiurile formate de aceste laturi în cele două triunghiuri sunt congruente.

Un poligon cu n laturi poate fi triangularizat în $n - 2$ triunghiuri. Dacă pentru două poligoane cele $n - 2$ triunghiuri sunt respectiv asemenea, atunci putem afirma că poligoanele sunt asemenea.

De aici putem deduce că două poligoane sunt asemenea dacă au laturile și diagonalele care pleacă dintr-un anumit vârf respectiv proporționale.

Vom rezolva problema folosind această proprietate. Vom considera vârfurile primului poligon în ordinea dată și vom încerca să verificăm dacă putem determina o ordine a vârfurilor celui de-al doilea poligon pentru care să obținem asemănarea.

Vom începe pe rând cu fiecare dintre vârfurile 1, ..., n. Pentru fiecare vârf k, vom considera pentru început ordinea k, $k + 1$, ..., n, 1, 2, ... $k - 1$ și apoi ordinea k, $k - 1$, ..., 1, n, $n - 1$, ..., $k + 1$.

Pentru a reduce puțin numărul calculelor vom lucra cu pătratele distanțelor în loc de distanțe propriu-zise. Acest lucru este posibil deoarece următoarea relație este întotdeauna adevărată:

$$\frac{a}{b}=\frac{c}{d}\Leftrightarrow\frac{a^2}{b^2}=\frac{c^2}{d^2}.$$

De asemenea, în loc de a testa o relație de tipul $a / b = c / d$, vom testa relația echivalentă $a \cdot d = b \cdot c$.

Vă prezentăm în continuare versiunea oficială a soluției pentru această problemă:

Au rezolvat corect:

Csaba Andras, Oradea
Ciprian Baicu, Dr. Turnu Severin
Vencel Bors, Oradea
Daniel Crișan, Râmnicu Vâlcea
Silvestru Negruseri, Bistrița
Csaba Patcas, Oradea
Dan Popovici, Arad
Radu Ștefan, Brașov

Listing POLY.CPP

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define epsilon 0.001

double x1[100],y1[100], // primul poligon
       x2[100],y2[100]; // al doilea poligon
int n, // numarul de varfuri
    c[100]; // ordinea considerata

void ReadData(void){
  FILE *f=fopen("POLY.IN","rt");
  fscanf(f,"%d",&n);
  for (int i=0;i<n;i++)
    fscanf(f,"%lf%lf",&x1[i],&y1[i]);
  for (i=0;i<n;i++)
    fscanf(f,"%lf%lf",&x2[i],&y2[i]);
}

inline double length1(int i,int j){
// lungimea unei laturi sau a unei diagonale
// in primul poligon
  return (x1[i]-x1[j])*(x1[i]-x1[j])+
         (y1[i]-y1[j])*(y1[i]-y1[j]);
}
```

```
inline double length2(int i,int j){
// lungimea unei laturi sau a unei diagonale
// in primul poligon
  return (x2[i]-x2[j])*(x2[i]-x2[j])+
         (y2[i]-y2[j])*(y2[i]-y2[j]);
}

int Ok(void){
// verificarea pastrarii raportului intre
// lungimile laturilor si diagonalelor
// corespunzatoare
  double denominator=length1(0,n-1),
         nominator=length2(c[0],c[n-1]),
         denominator2,
         nominator2;
  // s-a determinat raportul dintre
  // lungimile laturilor care unesc primul si
  // ultimul varf in cele doua poligoane

  // se verifica pastrarea raportului pentru
  // celelalte laturi
  for (int i=0;i<n-1;i++){
    denominator2=length1(i,i+1);
    nominator2=length2(c[i],c[i+1]);
    // se verifica pastrarea raportului
    if (fabs(denominator*nominator2-
        denominator2*nominator)>epsilon)
      // daca raportul nu se pastreaza
      // atunci nu am gasit ordinea pentru
      // care cele doua poligoane sunt
      // asemenea
      return 0;
  }

  // se verifica pastrarea raportului pentru
  // diagonalele care pleaca din primul varf
  // al celor doua poligoane
  for (i=0;i<n-1;i++){
    denominator2=length1(0,i);
    nominator2=length2(c[0],c[i]);
    // se verifica pastrarea raportului
    if (fabs(denominator*nominator2-
        denominator2*nominator)>epsilon)
      // daca raportul nu se pastreaza
      // atunci nu am gasit ordinea pentru
      // care cele doua poligoane sunt
      // asemenea
      return 0;
  }
  // raportul s-a pastrat, deci aceasta este
  // ordinea pentru care cele doua
  // poligoane sunt asemenea si constituie
  // solutia problemei
  return 1;
}
```

```
void WriteSolution(void){
  FILE *f=fopen("POLY.OUT","wt");
  for (int i=0;i<n;i++)
    // se aduna 1 deoarece am considerat ca
    // indicii incep de la valoarea
    // implicita (zero)
    fprintf(f,"%d\n",c[i]+1);
  fclose(f);
  // se intrerupe executia programului
  // pentru ca nu mai este nevoie sa gasim
  // alte solutii
  exit(1);
}

void SameOrder(int k){
// cele doua poligoane sunt parcurse in
// acelasi sens
  for (int i=k;i<n;i++)
    c[i-k]=i;
  for (i=0;i<k;i++)
    c[n-k+i]=i;
  if (Ok())
    WriteSolution();
}

void ReverseOrder(int k){
  for (int i=k;i>=0;i--)
    c[k-i]=i;
  for (i=n-1;i>k;i--)
    c[n-i+k]=i;
  if (Ok())
    WriteSolution();
}

void Check(void){
  for (int i=0;i<n;i++){
    // acelasi sens de parcurgere
    SameOrder(i);
    // sensuri diferite de parcurgere
    ReverseOrder(i);
  }
}

void NoSolution(void){
  FILE *f=fopen("POLY.OUT","wt");
  fprintf(f,"0\n");
  fclose(f);
}

void main(void){
  ReadData();
  Check();
  // nu am reusit sa gasim nici o solutie
  NoSolution();
}
```

Dreptunghi

Această problemă a fost rezolvată corect de 76 dintre cei care au participat la a doua rundă a *Concursului de Programare Agora*. Problema era foarte simplă, fiind o aplicație clasică a metodei programării dinamice.

Vom construi o matrice A de dimensiuni $m \times n$, elementul a_{ij} indicând numărul minim de pătrate în care poate fi împărțit un dreptunghi care are lungimile laturilor i și j. Este evident faptul că elementele a_{ii} au valoarea 1.

Pentru a determina valorile elementelor a_{ij} va trebui să determinăm poziția în care ar trebui să efectuăm tăietura pentru a minimiza numărul pătratelor în care va fi împărțit dreptunghiul care are laturi de lungimi i și j.

Pentru aceasta vom determina numărul de pătrate corespunzător fiecărei tăieturi verticale și fiecărei tăieturi orizontale. Formula pe care o vom folosi este:

$$a_{ij} = \min\left(\min_{k=1,i}(a_{kj} + a_{i-k,j}), \min_{k=1,j}(a_{ik} + a_{i,j-k})\right).$$

Se observă că numărul minim de pătrate în care poate fi împărțit un dreptunghi cu laturile i și j este egal cu numărul minim de pătrate în care poate fi împărțit orice dreptunghi cu laturile $k \cdot i$ și $k \cdot j$, pentru orice număr natural $k > 0$.

De aceea, pentru fiecare element al matricei A, am putea verifica dacă nu am determinat deja valoarea corespunzătoare. Valoarea a fost deja determinată numai dacă $\text{cmmdc}(i, j) > 1$. În acest caz avem $a_{ij} = a_{i/\text{cmmdc}(i,j),\, j/\text{cmmdc}(i,j)}$.

Prin aceasta timpul de execuție se reduce semnficativ. Totuși, o variantă de rezolvare care nu folosește această observație se încadrează fără probleme în timpul de execuție admis.

Varianta oficială a rezolvării acestei probleme este prezentată în continuare:

Listing RECT.CPP

```
#include <stdio.h>

int m,n,a[101][101];

void ReadData(void){
// citirea datelor de intrare
  FILE *f=fopen("RECT.IN","rt");
  fscanf(f,"%d%d",&m,&n);
  fclose(f);
}

int gcd(int m,int n){
// functie recursiva pentru determinarea
// celui mai mare divizor comun a doua
// numere naturale
  return (n)?gcd(n,m%n):m;
}

void FindSolution(void){
  int div;
  a[1][1]=1;
  // pentru lungimile laturilor 1 si 1 avem
  // un singur patrat
  for (int i=1;i<=m;i++)
    for (int j=1;j<=n;j++)
      if ((div=gcd(i,j))>1)
        a[i][j]=a[i/div][j/div];
      else{
        // numarul maxim de patrate in care
        // poate fi impartit un dreptunghi
        // cu laturile de lungimi i si j
        // este i * j
        a[i][j]=i*j;
        // taieturi orizontale
        for (int k=1;k<i;k++)
          if (a[k][j]+a[i-k][j]<a[i][j])
            a[i][j]=a[k][j]+a[i-k][j];
        // taieturi verticale
        for (k=1;k<j;k++)
          if (a[i][k]+a[i][j-k]<a[i][j])
            a[i][j]=a[i][k]+a[i][j-k];
      }
}

void WriteSolution(void){
  FILE *f=fopen("RECT.OUT","wt");
  fprintf(f,"%d",a[m][n]);
  fclose(f);
}

void main(void){
  ReadData();
  FindSolution();
  WriteSolution();
}
```

Au rezolvat corect:

Cristian Alexandrescu, Botoşani
Csaba Andras, Oradea
Mugurel Ionuţ Andreica, Bucureşti
Elvis Asaftei, Vaslui
Victor Asavei, Drobeta Turnu Severin
Cristian Băicoianu, Ploieşti
Ciprian Baicu, Drobeta Turnu Severin
Adrian Balasko, Zalău
Andrei Benea, Focşani
Valentin Bisa, Lugoj
Denis Bogdănaş, Iaşi
Daniel Bolohan, Iaşi
Vencel Bors, Oradea
Cătălin Bujdei, Braşov
Nadina Busuioc, Constanţa
Ciprian Cană, Vaslui
Adrian Cârcu, Bistriţa
Adrian Cearnău, Bucureşti
Bogdan Chirilă, Botoşani
Mihai-Silviu Chiru, Brăila
Horia Ciurdar, Timişoara
Alexandra-Elena Constantin, Ploieşti
Horea Coroiu, Cluj-Napoca
Daniel Crişan, Râmnicu Vâlcea
Andrei Csibi, Cluj-Napoca
Vlad Dascălu, Bacău
Andrei David, Râmnicu Vâlcea
Radu Dondera, Bucureşti
George Drumea, Bucureşti
Andrei-Liviu Dumbrava, Iaşi
Octavian-Daniel Dumitran, Bucureşti
Bogdan Dumitru, Ploieşti
Dan Ghinea, Bucureşti
Aurelian Ghiţă, Buzău
Andrei Giurgiu, Bucureşti
Claudiu Gruia, Bucureşti
Gabor Gyebnar, Ungaria
Mihai Hărănguş, Cluj-Napoca
Andrei Huţu, Bacău
Claudiu Ionesi, Bacău
Cătălin Kesa, Buşteni
Artur Kuczapski, Oradea
Liviu Lalescu, Craiova
Codruţ-Lucian Lazăr, Câmpeni
Bogdan Lucaciu, Drobeta Turnu Severin
Maximilian Machedon, Bucureşti
Paul Marinescu, Buzău
Andrei Markovits, Satu Mare
Mihail Piţ-Rada, Drobeta Turnu Severin
Petko Minkov, Bulgaria
Emilian Miron, Bacău
Alexandru Mosoi, Bacău
Petru-Simon Moţ, Brăila
Răzvan Muşaloiu-Elefteri, Bucureşti
Raluca Muşaloiu-Elefteri, Bucureşti
Cosmin-Silvestru Negruseri, Bistriţa
Bogdan Nicolae, Sibiu
Sorin Otescu, Braşov
Mihai-Vlad Pantiş, Cluj-Napoca
Gabriel Pârvan, Râmnicu Sărat
Flaviu Paşca, Bistriţa
Csaba Patcas, Oradea
Mihai Pătraşcu, Craiova
Liviu Păunescu, Constanţa
Valentin Pop, Carei
Mihail Popa, Bucureşti
Lucian-Raul Silistru, Vaslui
Bogdan Stan, Câmpina
Lucian-Daniel Stanciu, Brăila
Mihai Stoicescu, Boldeşti-Scăeni
Mihai Stroe, Bucureşti
Radu Ştefan, Braşov
Cristian Toth, Lugoj
Vlad Vâlceanu, Arad
Victor Vernescu, Constanţa
Violeta Voinescu, Târgu Jiu

Runda 6

Vă prezentăm în continuare cele trei probleme de la cea de-a patra rundă a Concursului de Programare Agora. Cei care doresc să participe numai la această ultimă rundă, pot trimite mesajul de înscriere până la data de 5 Aprilie 2001.

CPAR6P1: Inversiuni

Nume fișier sursă: `INVERS.PAS`, `INVERS.C` sau `INVERS.CPP`
Nume fișier de intrare: `INVERS.IN`
Nume fișier de ieșire: `INVERS.OUT`
Descriere: Se dă un vector a care conține n ($2 \le n \le 10.000$) elemente și se cere determinarea numărului de perechi (i, j) cu proprietatea $i < j$ și $a_i > a_j$.
Date de intrare: Fișierul de intrare va conține pe prima linie numărul n al elementelor vectorului. Pe următoarea linie se află cele n elemente ale vectorului, separate prin spații.
Date de ieșire: Fișierul de ieșire va conține numărul de perechi (i, j) care au proprietatea cerută.
Exemplu:

`INVERS.IN`	`INVERS.OUT`
`4`	`5`
`4 2 3 1`	

Timp de execuție: 1 secundă/test.

CPAR6P2: Rețea

Nume fișier sursă: `NET.PAS`, `NET.C` sau `NET.CPP`
Nume fișier de intrare: `NET.IN`
Nume fișier de ieșire: `NET.OUT`
Descriere: Se consideră o rețea pătratică cu n ($1 \le n \le 100$) linii și coloane. Din fiecare punct al rețelei se poate ajunge în unul din cei patru vecini de pe orizontală sau de pe verticală. Se consideră m ($1 \le m \le n^2$) puncte ale rețelei. Să se determine dacă există trasee distincte care pornesc din cele m puncte și ajung la marginea rețelei (pe prima/ultima linie/coloană).
Date de intrare: Prima linie a fișierului de intrare conține numărul n al liniilor și coloanelor rețelei și numărul m al punctelor de plecare. Următoarele m linii conțin coordonatele punctelor de plecare.
Date de ieșire: Dacă există m drumuri distincte până la marginea rețelei atunci fișierul de ieșire va descrie fiecare dintre aceste drumuri. Prima linie corespunzătoare descrierii unui traseu conține numărul k al vârfurilor din care este format traseul, iar a doua linie conține coordonatele punctelor de pe traseu cuprinse între paranteze. Dacă nu există trasee distincte, fișierul de ieșire va conține doar cifra 0.
Exemplu:

`NET.IN`	`NET.OUT`
`6 10`	`1`
`1 3`	`(1 3)`
`2 2`	`2`
`2 3`	`(2 2) (2 1)`
`2 4`	`4`
`4 2`	`(2 3) (3 3) (3 2) (3 1)`
`4 3`	`4`
`4 4`	`(2 4) (3 4) (3 5) (3 6)`
`6 2`	`2`
`6 3`	`(4 2) (4 1)`
`6 4`	`4`
	`(4 3) (5 3) (5 2) (5 1)`
	`3`
	`(4 4) (4 5) (4 6)`
	`1`
	`(6 2)`
	`1`
	`(6 3)`
	`1`
	`(6 4)`

Timp de execuție: 1 secundă/test.

CPAR6P3: Numere prime

Nume fișier sursă: `PRIME.PAS`, `PRIME.C` sau `PRIME.CPP`
Nume fișier de intrare: `PRIME.IN`
Nume fișier de ieșire: `PRIME.OUT`
Descriere: Se dă un număr natural n ($2 \le n \le 100$) și se cere determinarea a m ($1 \le m \le n$) numere prime care să aibă exact n cifre.
Date de intrare: Prima linie a fișierului de intrare conține numărul n al cifrelor numerelor prime care trebuie determinate și numărul m al numerelor prime care trebuie determinate.
Date de ieșire: Fișierul de ieșire va conține m numere prime care conțin exact n cifre, câte unul pe o linie.
Exemplu:

`PRIME.IN`	`PRIME.OUT`
`2 2`	`23`
	`29`

Timp de execuție: 1 secundă/test.

Ginfo Top 100

1. Maximilian Machedon, București	1171	6. Csaba Andras, Oradea	1099
2. George Drumea, București	1165	7. Mihai Stroe, București	1089
3. Ciprian Cană, Vaslui	1161	8. Bogdan Stan, Câmpina	1065
4. Radu Ștefan, Brașov	1124	9. Cristian Băicoianu, Ploiești	1062
5. Adrian Cârcu, Bistrița	1114	10. Gabriel Pârvan, Râmnicu Sărat	1034

11. Mugurel Ionuț Andreica, București	1029	41. Andrei Csibi, Cluj-Napoca	773	71. Lucian-Daniel Stanciu, Brăila	418
12. Liviu Lalescu, Craiova	1028	42. Horia Ciurdar, Timișoara	763	72. Bogdan Lucaciu, Dr. Turnu Severin	412
13. Daniel Crișan, Râmnicu Vâlcea	1015	42. Andrei David Râmnicu, Vâlcea	763	73. Paul Rusu, Cluj-Napoca	396
14. Dan Ghinea, București	1013	44. Alexandru Mosoi, Bacău	755	74. Andrei-Liviu Dumbrava, Iași	387
15. Ciprian Baicu, Drobeta Turnu Severin	1006	45. Cristian Toth, Lugoj	754	75. Ștefan Bucur, București	378
16. Denis Bogdănaș, Iași	1004	46. Valentin Bisa, Lugoj	719	76. Andrei Benea, Focșani	376
17. Cristian Alexandrescu, Botoșani	976	47. Mihai Hărănguș, Cluj-Napoca	718	77. Petko Minkov, Bulgaria	366
18. Claudiu Gruia, București	965	48. Răzvan Mușăloiu-Elefteri, București	706	78. Livia Bana, Drobeta Turnu Severin	359
19. Mihai Pătrașcu, Craiova	946	49. Cătălin Kesa, Bușteni	696	79. Aurel-Mihai Fulger, Constanța	357
20. Victor Asavei, Drobeta Turnu Severin	944	50. Paul Marinescu, Buzău	693	80. Cristian Bojinovici, București	353
21. Vlad Vâlceanu, Arad	918	51. Horea Coroiu, Cluj-Napoca	692	81. Bogdan-Vlad Țira, Cluj-Napoca	349
22. Liviu Păunescu, Constanța	916	52. Marius Andrei, București	658	82. Emilian Miron, Bacău	337
23. Octavian-Daniel Dumitran, București	914	52. Codruț-Lucian Lazăr, Câmpeni	658	83. Mihai Gojol, Cluj-Napoca	335
24. Bogdan Nicolae, Sibiu	907	54. Victor Vernescu, Constanța	655	84. Raluca Sauciuc, București	330
25. Vencel Bors, Oradea	900	55. Liviu-Cosmin Andreicuț, București	640	85. Lucian Burja, Blaj	320
25. Mihail Popa, București	900	56. Cosmin Răianu, Constanța	596	86. Costin Speciac, Buzău	316
25. Cosmin-Silvestru Negruseri, Bistrița	900	57. Radu Dondera, București	593	87. Nadina Busuioc, Constanța	315
28. M. A. Safari Ghahsareh, Iran	892	58. Artur Kuczapski, Oradea	591	88. Silviu-Gabriel Udrea, Târgu Jiu	305
29. Vincent Groenhuis, Olanda	842	59. Valentin Pop, Carei	577	89. Adrian Balasko, Zalău	304
30. Mihai-Vlad Pantiș, Cluj-Napoca	841	60. Andrei Giurgiu, București	570	90. Johannes Slotta, Germania	298
31. Radu Vătavu, Suceava	838	61. Adrian Cearnău, București	566	91. Bogdan-Nicolae Șanta, Cluj-Napoca	290
32. Vlad Dascălu, Bacău	832	62. Lucian-Raul Silistru, Vaslui	554	92. Daniel Comșa, Crăciunelu de Jos	286
33. Dan Popovici, Arad	827	63. Andrei Markovits, Satu Mare	546	93. Alina Fețnic, Cluj-Napoca	284
34. Csaba Patcas, Oradea	825	64. Mihail Pițˇ-Rada, Dr. Turnu Severin	545	94. Mihai-Silviu Chiru, Brăila	281
35. Ioana-Maria Ileană, Alba Iulia	824	65. Bogdan-Milovan Piloga, Timișoara	489	95. Daniel Bolohan, Iași	266
36. Bogdan Dumitru, Ploiești	811	66. Arcadie Crăcan, Iași	456	95. Preslav Nakov, Bulgaria	266
37. Alexandra-Elena Constantin, Ploiești	808	67. Raluca Mușăloiu-Elefteri, București	443	97. Narayanan Arvind, India	264
38. Flaviu Pașca, Bistrița	802	68. Petru-Simon Moț, Brăila	437	97. Radu Cocieru, Sibiu	264
39. Aurelian Ghiță, Buzău	785	69. V. Muthuramakrishnan, India	421	99. Constantin Asofiei, Târgu Neamț	251
40. Sorin Otescu, Brașov	784	70. Laurent Demonet, Franța	420	100. Cristian Ioniță, București	243

Arbori BINARI de formă ARBITRARĂ

Adrian Vasilescu

Acest articol prezintă modul în care pot fi efectuate eficient diferite operații folosind arborii binari. Vor fi prezentate diverse strategii cu ajutorul cărora se pot îmbunătăți performanțele algoritmilor care folosesc această structură de date și care, se pare, mai au încă multe "secrete".

Introducere

Procesarea eficientă a datelor reprezintă una dintre preocupările majore din domeniul programării si administrării structurilor de date, în mod special în cadrul aplicațiilor în care volumul de informație prelucrată și stocată este ridicat.

În acest context, căutarea si localizarea eficientă de date prezintă un interes aparte, știut fiind faptul că aceste operații sunt costisitoare din punct de vedere computațional și în anumite cazuri chiar prohibite din cauza accesului lent pe disc. Astfel, o căutare de informații într-o structură de date ar putea necesita vizitarea și verificarea identității unui volum foarte mare de informație, această operație devenind cu ușurință o redutabilă sursă de ineficiență computațională.

Procesul de căutare poate fi însă îmbunătățit prin organizarea informației în structura de date, astfel încât lungimea traseelor de căutare să fie drastic redusă și optimizată cât mai mult posibil. Sunt cunoscute diferite strategii de îmbunătățire a performanței de căutare în diferite tipuri de structuri de date, strategii ce speculează în mod eficient particularitățile specifice ale aplicației considerate [1].

În cazul structurilor de date pentru care ordinea de introducere a datelor nu este cunoscută apriori și în care aceste date nu pot fi ordonate în vreun fel, structura care oferă un mecanism de căutare rapidă este arborele binar echilibrat (balansat).

În acest articol sunt prezentate considerentele teoretice si detaliile de programare ale acestui tip de arbore binar de formă arbitrară în cazul cărora inserarea de date se face pe baza unor numere de prioritate obținute de la un generator de numere arbitrare.

Accentul este pus pe implementarea modulară a structurii de *arbore binar de formă arbitrară* într-o manieră ce exclude accesul la date bazat pe indexare cât și pe utilizarea blocurilor de memorie continuă și mărime fixă, așa numiții *arrays* care nu permit alocarea dinamică a spațiului de memorie.

În încheierea articolului, o analiză a complexității algoritmului va demonstra propietatea arborelui binar de formă arbitrară de a fi echilibrat.

Structura de arbore binar

Arborele binar este o structură de date multinivel, reprezentată de o colecție de noduri, având un nod *rădăcină* și doi subarbori, unul *stâng* și altul *drept*, conectați la nodul rădăcină. Astfel, arborele binar reprezentat în figura 1a) este compus din patru noduri interne (reprezentate prin cercuri) și cinci noduri externe (reprezentate prin pătrate), aceștia din urmă corespunzând unor subarbori vizi. În figura 1b), nodurile externe nu au mai fost reprezentate, în general aceste noduri fiind lipsite de o reală importanță practică sunt ocupate de pointerul `NULL`, pentru care nu este alocat spațiu de memorie.

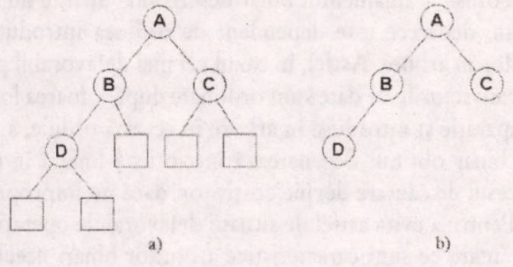


Figura 1. Structura de arbore binar

O clasificare și mai convențională a nodurilor într-un arbore se poate obține pe bază genealogică. Astfel, nodul *A* din figura 1 este ***ancestor*** pentru toate nodurile situate în subarborii săi care sunt ***descendenți*** ai săi. De asemenea, nodul *A* este ***părinte*** pentru nodurile situate pe nivelul ierarhic următor, coborând în arbore, și care sunt legate direct de nodul *A*. În figura 1, nodurile *B* și *C* sunt ***fii*** ai nodului părinte *A*.

Este de notat faptul că traseul de la un nod ancestor n_1 către descendenții săi n_k este unic, traseul fiind determinat de secvența de noduri unică $n_1, n_2, ..., n_k$ în care nodul n_i este părinte al nodului n_{i+1} pentru $i = 1, 2, ..., k - 1$. Prin urmare, lungimea traseului de vizitare este determinată de numărul de legături dintre nodurile aparținând traseului ($k - 1$ pentru nodul n_k). În figura 1a), traseul de la *A* la *D* are lungimea 2, fiind compus din secvența de noduri *A-B-D*. Poziționarea unui nod în arbore se face prin definirea a doi scalari complementari denumiți în mod sugestiv ***adâncimea*** și ***înălțimea*** nodului în arbore. Astfel, adâncimea nodului n_k corespunde lungimii traseului de la rădăcina arborelui până la nod care este $k - 1$, în timp ce înălțimea aceluiași nod n_k este reprezentată de lungimea maximă a traseului având ca rădăcină nodul însuși și terminând la nodul extern cel mai îndepărtat în subarbore.

Căutarea și localizarea datelor

Pentru ca operațiile de căutare și localizare să fie eficiente, datele în arborele binar trebuie aranjate în așa fel încât lungimea traseelor de căutare sa fie cât mai mică posibil, lucru ce poate fi realizat prin gruparea nodurilor pe niveluri ierarhice pe cât posibil mai apropiate de rădăcina arborelui.

În general, inserarea seturilor de date în arbore se face pe baza unui criteriu de comparație binar, astfel încât seturile de date ale căror valori de comparație respectă criteriul impus să fie direcționate în subarborii stângi. Similar, seturile de date ale căror valori de comparație nu satisfac criteriul impus sunt direcționate în subarborii drepți, în final setul de date introdus în arbore ocupând poziția unui nod extern.

Este de remarcat faptul că invocarea criteriului de comparație se face pentru fiecare nod din traseu, aceasta constituind o sursă de calcul intensiv. În acest aranjament al datelor, efortul de căutare și localizare este considerabil redus prin faptul că aceste operațiuni sunt efectuate numai într-un anumit sector al arborelui, în subarborii succesivi stângi sau drepți, după caz.

Totuși, aranjamentul binar descris mai înainte nu este optim, deoarece este dependent de ordinea introducerii datelor în arbore. Astfel, în cazul cel mai defavorabil pentru care seturile de date sunt ordonate după valoarea lor de comparație și introduse în arbore în această ordine, arborele binar obținut degenerează într-o listă liniară în care procesul de căutare devine costisitor, dacă nu impropriu.

Pentru a evita astfel de situații defavorabile operațiilor de căutare ce sunt caracteristice arborilor binari neechilibrați sau nebalansați, se impune ca inserarea seturilor de date să se efectueze în așa fel încât procesul de formare al arborelui să devină independent de ordinea seturilor de date pe baza cărora se construiește arborele.

Prin această strategie se urmărește ca adâncimea arborelui să fie drastic redusă, prin aceasta micșorându-se lungimea traseelor de căutare în arbore și prin urmare a efortului computațional de căutare de date. Arborele binar este considerat echilibrat sau balansat dacă lungimile traseelor de la rădăcină până la nodurile externe sunt aproximativ egale. În cazul ideal, traseele complete au lungimi egale și arborele binar este considerat perfect echilibrat.

Construirea unui arbore binar

O metodă eficientă care asigură formarea arborelui în totală independență de ordinea de introducere a seturilor de date este aceea de a utiliza numere arbitrare ca valori de comparație necesare construcției arborelui. Aceste numere arbitrare constituie prioritățile de intrare atribuite seturilor de date și pot fi obținute de la generatorul de numere arbitrare al compilatorului folosit în prelucrarea codului sursă.

Astfel, fiecărui set de date i se atribuie un număr de prioritate arbitrar pozitiv și subunitar, introducerea datelor în arbore efectuându-se pe baza numerelor de prioritate atribuite în locul valorilor de comparație și folosind același criteriu de comparație binar, astfel încât prioritatea unui set de date să nu fie mai mare decât prioritățile seturilor de date descendente.

În acest mod, regula de bază a arborelui binar rămâne în efect, doar forma sa fiind arbitrară și controlată de numerele de prioritate arbitrare. Din acest motiv, arborele binar descris mai înainte este denumit, în mod sugestiv, ***arbore binar de formă arbitrară***.

Pentru ca acest tip de arbore să fie robust este suficient ca numerele de prioritate arbitrare să fie unice, iar ordinea lor de generare să fie neliniară și aciclică.

Din acest punct de vedere, generatorul de numere `rand()` al compilatorului C++ corespunde scopului arborelui binar de formă arbitrară.

Informația care trebuie stocată este organizată în blocuri de memorie, așa-numitele obiecte care la rândul lor sunt instanțe ale claselor prototip, care nu sunt altceva decât șabloane special proiectate în cadrul aplicației curente după care sunt generate obiectele propriu-zise.

Așadar, o structură de date poate fi concepută ca o colecție de blocuri de memorie pentru stocarea informațiilor și un set de operațiuni special proiectate pentru inserarea, modificarea și accesarea informației stocate.

În acest mod, separarea algoritmului de structura sa de date facilitează un nivel înalt de abstractizare în proiectarea și implementarea eficientă a oricărei operațiuni în cadrul aplicației. Acest proces stă la baza programării modulare, exprimată în termeni de clase prototip și instanțe ale acestora (obiecte), și al cărei scop principal constă în reducerea complexității de exprimare a algoritmilor pe bază de eficiență computațională.

Avantajele programării modulare sunt diverse și incontestabile, conceptul de clasă prototip oferind un înalt grad de reutilizare a lor în cadrul altor aplicații similare cu un efort de programare minim. De asemenea, clasele prototip pot fi extinse sau alterate fără a fi necesară intervenția în structura internă a programului sursă. Astfel, unul din scopurile principale ale activității de producere de *software*, acela de a permite extinderea și menținerea operativă a programelor pe perioade lungi de timp, este îndeplinit.

Primul pas în implementarea arborelui binar de formă arbitrară este de a defini un set abstract de date format din clase prototip cu ajutorul cărora structura de arbore să fie exprimată mai mult la nivel conceptual și mai puțin în termeni referitori la detalii de programare.

Procesul de identificare și formulare a claselor prototip este complex și de obicei repetitiv necesitând câteva cicluri de analiză profundă a detaliilor problemei și formulare ierarhică a algoritmilor.

În același timp, trebuie avut în vedere și aspectul eficienței computaționale care, ignorat, poate duce la dificultăți neașteptate în extinderea și exploatarea economică a programului.

O fragmentare excesivă a codului sursă în obiecte conduce inevitabil la un mare număr de apeluri de funcții și o organizare internă a programului deosebit de complexă, acestea fiind recunoscute ca veritabile surse de ineficiență.

Așadar, procesul de proiectare orientată-obiect și implementare eficientă a modelelor computaționale este unul complex și de durată, necesitând multiple revizuiri și completări.

În ceea ce privește implementarea arborelui binar de formă arbitrară descris anterior, aspectul eficienței computaționale a fost compromis într-o foarte mică măsură numai pentru a oferi o mai bună claritate codului sursă, acesta fiind și principalul scop al actualei versiuni.

O primă entitate abstractă ce poate fi ușor identificată este aceea de ***nod*** al arborelui. Astfel, este creată o clasă denumită `TreeNode`, conținând un element membru denumit `value` și doi pointeri denumiți `_lchild` și `_rchild`, care reprezintă legăturile nodului către cei doi descendenți ai săi.

Prin intermediul pointerilor `_lchild` și `_rchild`, legăturile dintre nodurile arborelui sunt stabilite în mod automat de către constructorul clasei prototip `TreeNode` fără a necesita nici un fel de operații de indexare sau tabelare.

Destructorul `~TreeNode()` este responsabil pentru eliberarea spațiului de memorie ocupat de obiectul nod cât și de descendenții acestuia.

Pentru a asigura utilizarea eficientă a arborelui, clasele prototip vor fi implementate prin intermediul mecanismului `template<>`, astfel încât structura de arbore să devină independentă de tipul de date pe care va trebui să le înmagazineze.

Astfel, tipul datelor `T` care vor fi inserate în arbore este stabilit la inițializarea arborelui prin specificarea sa între parantezele unghiulare ale specificatorului `template<>`.

Acest tip `T` al datelor este la rândul său reprezentat de o clasă prototip special proiectată în cadrul aplicației curente.

Reprezentarea clasei prototip `TreeNode` în cod C++ cuprinde două părți. Prima este una declarativă și constituie interfața clasei care este implementată într-un fișier purtând denumirea clasei și având extensia **`.H`** sau **`.HPP`**.

```
template <class T> class TreeNode{
  protected:
    T value;
    TreeNode *_lchild;
    TreeNode *_rchild;
  public:
    TreeNode(T);
    virtual ~TreeNode(void);
};
```

Cea de-a doua parte a clasei constă în definirea propriu-zisă a metodelor de operare, această parte fiind implementată într-un fișier purtând denumirea clasei și având extensia **`.C`** sau **`.CPP`**.

Constructorul `TreeNode(T)` creează un arbore binar având un singur nod și două legături către fiii săi, fiecare fiind reprezentat de pointerul `NULL`.

```
template<class T>
TreeNode<T>::TreeNode(T value) :
    value(value),_lchild(NULL),_rchild(NULL){}
```

Destructorul clasei, `~TreeNode(void)` eliberează memoria ocupată de obiectul nod cât și pe cea a fiilor săi, dacă această memorie a fost alocată:

```
template<class T>
TreeNode<T>::~TreeNode(void){
   if (_lchild) delete _lchild;
   if (_rchild) delete _rchild;
}
```

Colecția de noduri aparținând arborelui va fi stocată într-o listă ordonată. Deoarece o altă operație foarte importantă ce trebuie implementată se referă la localizarea succesorului sau a predecesorului unui nod arbitrar din arbore, vom păstra și o listă care conține pointeri la nodurile învecinate din arbore. Astfel, vom introduce o nouă clasă prototip denumită `Node` conținând legăturile la nodurile succesor și predecesor ale nodului curent.

```
class Node{
  protected:
    Node *_next;
    Node *_prev;
  public:
    Node(void);
    virtual ~Node(void);
```

```
    inline Node *next(void) const;
    inline Node *prev(void) const;
    Node *remove(void);
    Node *insert(Node *newNode);
};
```

Constructorul `Node(void)` creează un nou tip de nod corespunzând unui element în listă:

```
Node::Node(void){
  _prev=this;
  _next=this;
}
```

Am folosit pointerul C++ **this** care indică obiectul element al listei. Acest constructor inițializează pointerii `_prev` și `_next`, reprezentând legăturile către nodurile predecesor și succesor din arbore cu pointerul **this**, urmând ca aceste legături să fie setate în mod corespunzător la timpul potrivit.

Destructorul `~Node(void)` este responsabil pentru eliberarea spațiului de memorie ocupat de obiectele tip `Node` după ce durata lor de viață se încheie. Acest destructor este declarat virtual pentru a asigura eliberarea de memorie în mod corect și pentru obiectele derivate de la această clasă prototip.

```
Node::~Node(void){}
```

Funcțiile membru `next()` și `prev()` sunt folosite pentru a obține adresele nodurilor succesor și predecesor.

Aceste două funcții sunt declarate **inline** din motive de eficiență, directiva **inline** sugerând compilatorului să introducă instrucțiunile din corpul funcției în codul sursă fără a mai utiliza mecanismul de activare a funcției, mecanism care este costisitor și nejustificabil în acest caz.

Pe de altă parte, membrii clasei `Node`, `_next` și `_prev`, nu trebuie alterați în nici un fel pe durata de viață a arborelui, motiv pentru care funcțiile `next()` și `prev()` vor fi declarate **const**.

```
inline Node *Node::next(void) const{
  return _next;
}
inline Node *Node::prev(void) const{
  return _prev;
}
```

Funcția membru `insert(Node *n)` inserează pointerul nodului n în listă imediat după nodul curent reprezentat de pointerul **this**, refacându-se în același timp legăturile dintre noduri:

```
Node *Node::insert(Node *node){
  Node *new_node = _next;
  node->_next = new_node;
  node->_prev = this;
  _next = node;
  new_node->_prev = node;
  return node;
}
```

Un nod din listă poate fi înlăturat prin intermediul funcției membru `remove(void)` astfel încât legăturile dintre nodurile rămase să poată fi refăcute:

```
Node *Node::remove(void){
  _prev->_next = _next;
  _next->_prev = _prev;
  _next = _prev = this;
  return this;
}
```

Pe baza celor două clase prototip `TreeNode` și `Node` definite mai înainte, este introdusă o nouă clasă cu ajutorul mecanismului de moștenire, clasă numită, în mod sugestiv, `BreidedNode`. Această nouă clasă permite accesul la nodurile arborelui pe baza relației genealogice părinte-descendenți dar și prin parcurgerea listei de noduri prin intermediul listei.

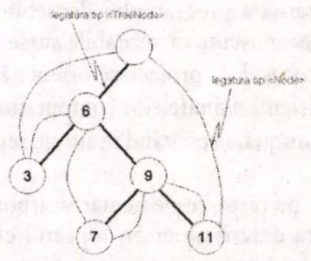


Figura 2. Arbore binar legat

```
template<class T>
class BraidedNode:public Node,
                  public TreeNode<T>{
public:
    BraidedNode( T );
    inline BraidedNode<T> *rchild(void) const;
    inline BraidedNode<T> *lchild(void) const;
    inline BraidedNode<T> *prev(void) const;
    inline BraidedNode<T> *next(void) const;
};
```

Constructorul `BreidedNode(T)` inițializează în mod explicit membrii clasei `TreeNode`:

```
template<class T>
BraidedNode<T>::BraidedNode(T value):
                        TreeNode<T>(value){}
```

Funcțiile membru `rchild()`, `lchild()`, `next()` și `prev()` returnează cele patru legături ale nodului, primele două aparținând structurii de arbore, iar ultimele două structurii de listă.

Aceste funcții sunt declarate **inline** și **const** din aceleași motive prezentate anterior.

```
template<class T>
inline BraidedNode<T> *BraidedNode<T>::
                          rchild(void) const{
  return (BraidedNode<T> *)_rchild;
}

template<class T>
inline BraidedNode<T> *BraidedNode<T>::
                          lchild(void) const{
  return (BraidedNode<T> *)_lchild;
}

template<class T>
inline BraidedNode<T> *BraidedNode<T>::
                          prev(void) const{
  return (BraidedNode<T> *)_prev;
}

template<class T>
inline BraidedNode<T> *BraidedNode<T>::
                          next(void) const{
  return (BraidedNode<T> *)_next;
}
```

Având structura de arbore deja conturată prin intermediul claselor prototip `TreeNode`, `Node` și `BraidedNode`, nu mai rămâne decât fixarea structurii sale funcționale reprezentată în principal de operațiile de inserare, căutare și localizare a datelor.

Procesul de inserare a datelor în arbore are loc în două faze distincte, mai întâi nodul este introdus în arbore prin metoda clasică (înlocuirea unui nod extern), după care se aplică o reordonare a nodurilor în arbore în funcție de numărul de prioritate al fiecăruia.

Trebuie menționat faptul că operația de reordonare necesită introducerea a două noi funcții `rotateRight()` și `rotateLeft()` în vederea poziționării nodurilor în arbore astfel încât nodul părinte să aibă numărul de prioritate mai mic decât al tuturor descendenților săi.

Astfel, dacă x este fiul stâng al nodului părinte y și are numărul de prioritate mai mic decât numărul de prioritate al nodului părinte y, atunci funcția `rotateRight()` va muta nodul x pe un nivel ierarhic superior nodului y în arbore, conform principiului de bază al arborelui binar de formă arbitrară.

De asemenea, este posibil ca acest principiu să nu fie respectat în cazul noului părinte al nodului x în noua sa configurație, caz în care va fi efectuată o nouă rotație a nodului părinte, o rotație de dreapta, dacă x este fiu stâng sau o rotație de stânga, dacă x este fiu drept.

Acest proces de rotație a nodurilor este continuat până când numărul de prioritate al nodului x va fi mai mic decât al tuturor nodurilor sale ascendente.

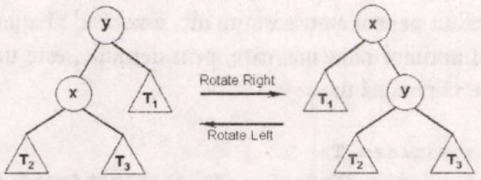


Figura 3. Rotații de noduri

Pentru a implementa funcțiile de rotație, este introdusă o nouă clasă prototip numită `RandomizedNode` al cărei acces la datele clasei `BreidedNode` este permis prin mecanismul de moștenire. Această nouă clasă conține pointerul care indică nodul părinte al nodului curent (`_parent`) cât și numărul său de prioritate (`_priority`).

```
template<class T>
class RandomizedNode: public BraidedNode<T>{
  protected:
    RandomizedNode *_parent;
    double _priority;
    void rotateRight(void);
    void rotateLeft(void);
    void bubbleUp(void);
    void bubbleDown(void);
  public:
    RandomizedNode(T value, int seed = -1 );
    inline RandomizedNode *lchild(void) const;
    inline RandomizedNode *rchild(void) const;
    inline RandomizedNode *prev(void) const;
    inline RandomizedNode *next(void) const;
    inline RandomizedNode *parent(void) const;
    inline double priority(void) const;
    friend class RandomizedSearchTree<T>;
};
```

Constructorul `RandomizedNode(T value, int seed = -1)` atribuie unui nou nod creat un număr de prioritate arbitrar, generat de funcția standard `rand()` a compilatorului C++.

```
template<class T>
RandomizedNode<T>::RandomizedNode(T value,
                int seed):BraidedNode<T>(value){
  if (seed != -1) srand(seed);
  _priority = (rand()) / (32767.0 + 1.0);
  _parent = NULL;
}
```

Funcțiile membru `rotateRight()` și `rotateLeft()` vor schimba forma arborelui prin actualizarea legăturilor nodurilor participante conform principiului reprezentat în figura 3). Astfel, prin rotația de dreapta, legătura nodului y la fiul său stâng este conectată la subarborele `T2` iar legătura nodului x la fiul său drept este conectată la nodul y. De asemenea, trebuie să verificăm dacă părintele nodului x

este definit pentru a nu permite nici unui nod să ajungă la nivelul nodului rădăcină, care, prin definiție, este unic și trebuie să rămână unic.

```
template<class T>
void RandomizedNode<T>::rotateRight(void){
  RandomizedNode<T> *y = this;
  RandomizedNode<T> *x = y->lchild();
  RandomizedNode<T> *p = y->parent();
  y->_lchild = x->rchild();
  if (y->lchild() != NULL)
    y->lchild()->_parent = y;
  if (p->rchild() == y) p->_rchild = x;
  else p->_lchild = x;
  x->_parent = p;
  x->_rchild = y;
  y->_parent = x;
  return;
}
```

Funcția `rotateLeft()` efectuează operația de rotație inversă, aceasta fiind exprimată în mod simetric funcției `rotateRight()` după cum urmează:

```
template<class T>
void RandomizedNode<T>::rotateLeft(void){
  RandomizedNode<T> *x = this;
  RandomizedNode<T> *y = x->rchild();
  RandomizedNode<T> *p = x->parent();
  x->_rchild = y->lchild();
  if ( x->rchild() != NULL )
    x->rchild()->_parent = x;
  if ( p->lchild() == x ) p->_lchild = y;
  else p->_rchild = y;
  y->_parent = p;
  y->_lchild = x;
  x->_parent = y;
  return;
}
```

Funcțiile `bubbleUp()` și `bubbleDown()` au scopul de a efectua rotațiile respective până când numărul de prioritate al nodului curent satisface principiul de construcție a arborelui binar de formă arbitrară. Operațiile de rotație sunt aplicate nodului părinte.

```
template<class T>
void RandomizedNode<T>::bubbleUp(void){
  RandomizedNode<T> *p = parent();
  if (priority() < p->priority()){
    if ( p->lchild() == this)
      p->rotateRight();
    else p->rotateLeft();
    bubbleUp();
  }
}
```

```
template<class T>
void RandomizedNode<T>::bubbleDown(void){
  double lcPriority = lchild()?
                      lchild()->priority():2.0;
  double rcPriority = rchild()?
                      rchild()->priority():2.0;
  double minPriority = (lcPriority <
          rcPriority)?lcPriority:rcPriority;
  if (priority() <= minPriority)
    return;
  if (lcPriority < rcPriority)
    rotateRight();
  else rotateLeft();
  bubbleDown();
  return;
}
```

Sensul operației de rotație este determinat de relația dintre nodurile fii, dacă fiul stâng are numărul de prioritate mai mic, o rotație de dreapta `rotateRight()` va muta fiul stâng pe un nivel superior în arbore, în timp ce nodul curent va fi mutat pe un nivel inferior, după cum se poate observa în figura 3).

În caz contrar, va fi aplicată o rotație de stânga `rotateLeft()`. Pentru a preveni orice interferență a nodurilor exterioare cu nodurile arborelui, nodurilor exterioare li se atribuie un număr de prioritate `2.0` sau cel puțin un număr din afara intervalului (`0.0`, `1.0`).

Funcțiile `rchild()`, `lchild()`, `next()`, `prev()` și `parent()` au fost implementate în scopul obținerii accesului la nodurile respective, acces foarte util în operațiile cu arbori.

```
template<class T>
inline RandomizedNode<T>
      *RandomizedNode<T>::lchild(void) const{
  return (RandomizedNode<T> *) _lchild;
}
```

```
template<class T>
inline RandomizedNode<T>
      *RandomizedNode<T>::rchild(void) const{
  return (RandomizedNode<T> *) _rchild;
}
```

```
template<class T>
inline RandomizedNode<T>
       *RandomizedNode<T>::prev(void) const{
  return (RandomizedNode<T> *) _prev;
}
```

```
template<class T>
inline RandomizedNode<T>
       *RandomizedNode<T>::next(void) const{
  return (RandomizedNode<T> *) _next;
}
```

```
template<class T>
inline RandomizedNode<T>
      *RandomizedNode<T>::parent(void) const{
  return (RandomizedNode<T> *) _parent;
}

template<class T>
inline double RandomizedNode<T>::
                          priority(void) const{
  return _priority;
}
```

Arborele binar propriu-zis este reprezentat de o nouă clasă prototip denumită `RandomizedSearchTree`. Membrii clasei sunt rădăcina arborelui (`root`) și un cursor (`cursor`).

O metodă foarte importantă a acestei clase este funcția de comparare `comparison(T,T)` pe baza căreia se face introducerea datelor în arbore.

Alte metode ale clasei sunt implementate pentru a facilita folosirea structurii de arbore în mod eficient.

```
template<class T> class RandomizedSearchTree{
  private:
    RandomizedNode<T> *root;
    RandomizedNode<T> *cursor;
    int (*comparison)(T,T);
    void _remove(RandomizedNode<T> *);
  public:
    RandomizedSearchTree(int (*)(T,T),
                         int seed = -1);
    ~RandomizedSearchTree(void);
    inline T next(void);
    inline T prev(void);
    inline T value(void);
    inline bool isFirst(void) const;
    inline bool isLast(void) const;
    inline bool isHead(void) const;
    inline bool isEmpty(void) const;
    inline T findMin(void);
    inline T findMax(void);
    T find(T);
    T locate(T);
    T insert(T);
    T remove(T);
    T removeMin(void);
    T removeMax(void);
    void remove(void);
    void inorder(void (*visit)(T));
};
```

Constructorul clasei, `RandomizedSearchTree(int (*)(T,T), int seed = -1)` inițializează un arbore binar de formă arbitrară care conține un singur nod izolat (`root`) care reprezintă radacina arborelui și al cărui număr de prioritate este `-1.0`.

```
template<class T>
RandomizedSearchTree<T>::RandomizedSearchTree
 (int (*cmp)(T,T),int seed): comparison(cmp){
  cursor = root = new RandomizedNode<T>
                                  (NULL,seed);
  root->_priority = -1.0;
}
```

Destructorul `~RandomizedSearchTree()` este responsabil pentru eliberarea spațiului de memorie a arborelui când durata sa de viață ia sfârșit.

```
template<class T>
RandomizedSearchTree<T>::
                  ~RandomizedSearchTree(void){
  delete root;
}
```

Funcțiile `next()`, `prev()`, `value()`, `isFirst()`, `isLast()`, `isHead()` și `isEmpty()` sunt definite după cum urmează:

```
template<class T>
inline T RandomizedSearchTree<T>::next(void){
  cursor = cursor->next();
  return cursor->value;
}

template<class T>
inline T RandomizedSearchTree<T>::prev(void){
  cursor = cursor->prev();
  return cursor->value;
}

template<class T>
inline T RandomizedSearchTree<T>::value(void){
  return cursor->value;
}

template<class T>
inline bool RandomizedSearchTree<T>::
                          isFirst(void) const{
  return (cursor == root->next()) &&
         (root != root->next());
}

template<class T>
inline bool RandomizedSearchTree<T>::
                          isLast(void) const{
  return (cursor == root->prev()) &&
         (root != root->next());
}

template<class T>
inline bool RandomizedSearchTree<T>::
                          isHead(void) const{
  return cursor == root;
}
```

```
template<class T>
inline bool RandomizedSearchTree<T>::
                                isEmpty(void) const{
  return root == root->next();
}
```

Funcția membru `insert(T avalue)` este responsabilă pentru introducerea datelor în arbore:

```
template<class T>
T RandomizedSearchTree<T>::insert(T avalue){
                          // find external node
  int result = 1;
  RandomizedNode<T> *p = root;
  RandomizedNode<T> *node = root->rchild();
  while (node){
    p = node;
    result = (*comparison)(avalue,p->value);
    if (result < 0)
      node = p->lchild();
    else
      if (result > 0)
        node = p->rchild();
      else
        return NULL;
  }
  cursor = new RandomizedNode<T>(avalue);
  cursor->_parent = p;
  if (result < 0){
     p->_lchild = cursor;
     p->prev()->Node::insert(cursor);
  }
  else{
    p->_rchild = cursor;
    p->Node::insert(cursor);
  }
     // bubble item according to its priority
  cursor->bubbleUp();
  return avalue;
}
```

Mecanismul de inserare a datelor este compus din două faze distincte. Într-o primă etapă, inserarea unei date x_i se efectuează prin plasarea sa în nodul extern corespunzător, făcând abstracție de numărul de prioritate atribuit. În etapa a doua, forma arborelui este ajustată corespunzător regulii priorităților în arbore, astfel încât prioritatea oricărui nod x_i din arbore să fie mai mare sau egală cu prioritatea părintelui său.

Funcția membru `find(T avalue)` este folosită în operația de căutare a datei `avalue` al cărei tip este indicat de variabila `T` care este identică cu cea a datelor stocate în arbore.

```
template<class T>
T RandomizedSearchTree<T>::find(T avalue){
  RandomizedNode<T> *node = root->rchild();
  while (node){
    int result = (*comparison)(avalue,
                                node->value);
    if (result < 0)
      node = node->lchild();
    else if (result > 0)
           node = node->lchild();
         else{
           cursor = node;
           return node->value;
         }
  }
  return NULL;
}
```

Operația de căutare a datei `avalue` se desfășoară de-a lungul unui traseu de căutare unic determinat începând de la rădăcină și coborând în arbore.

Pentru fiecare nod de pe traseu, este folosită funcția de comparație a arborelui pentru compararea datei căutate `avalue` cu cea stocată în nodul curent (`node->value`). Dacă `avalue < node->value`, procesul de căutare este continuat în subarborele stâng prin `node->_lchild`, în caz contrar procesul de căutare fiind continuat în subarborele drept. În cele din urmă, cursorul arborelui este fixat la nodul datei găsite și funcția de căutare se termină prin returnarea datei găsite. Dacă data căutată nu se află în arbore, se returnează pointerul `NULL`.

O altă operație de căutare este reprezentată de funcția membru `locate(T avalue)` care, aplicată argumentului său, returnează argumentul însuși dacă această dată este conținută în arbore. În cazul în care data nu este conținută în arbore, se retunează data cu valoarea imediat următoare în ordine descrescătoare.

```
template<class T>
T RandomizedSearchTree<T>::locate(T avalue){
  RandomizedNode<T> *locate = root;
  RandomizedNode<T> *node = root->rchild();
  while (node){
    int result = (*comparison)(avalue,
                                node->value);
    if (result < 0) node = node->lchild();
    else
      if (result > 0){
        locate = node;
        node = node->rchild();
      }
      else{
        cursor = node;
        return cursor->value;
      }
  }
  cursor = locate;
  return cursor->value;
}
```

Funcțiile membru `findMin()` și `findMax()` sunt responsabile pentru returnarea datelor considerate minime, respectiv maxime potrivit criteriului de comparație impus. Aceste funcții sunt extrem de eficiente având în vedere structura ordonată a arborelui, nici o operație de căutare nefiind necesară; aceste funcții sunt declarate **inline**.

```
template<class T>
inline T RandomizedSearchTree<T>::
                                findMin(void){
  cursor = root->next();
  return cursor->value;
}

template<class T>
inline T RandomizedSearchTree<T>::
                                findMax(void){
  cursor = root->prev();
  return cursor->value;
}
```

Funcția `remove(T avalue)` șterge data `avalue` din arbore dacă aceasta există, cursorul arborelui fiind mutat la locația predecesorului datei `avalue`. Implementarea acestei funcții se bazează pe funcția membru `_return(RandomizedNode<T> *node)` declarată **private** în clasa `RandomizedSearchTree`.

```
template<class T>
T RandomizedSearchTree<T>::remove(T avalue){
  T value = find(avalue);
  if (value){
    remove();
    return value;
  }
  return NULL;
}

template<class T>
void RandomizedSearchTree<T>::
          _remove(RandomizedNode<T> *node){
  node->_priority = 1.5;
  node->bubbleDown();
  RandomizedNode<T> *p = node->parent();
  if (p->lchild() == node) p->_lchild = NULL;
  else p->_rchild = NULL;
  if (cursor == node) cursor = node->prev();
  node->Node::remove();
  delete node;
  return;
}
template<class T>
void RandomizedSearchTree<T>::remove(void){
  if (cursor != root) _remove( cursor );
  return;
}
```

Analiza performanței

În cazul arborelui binar de formă arbitrară, o analiză a performanței se poate efectua numai statistic, dat fiind faptul că forma arborelui este controlată de numerele de prioritate arbitrare asociate seturilor de date. Principalul obiectiv al analizei de față îl constituie determinarea numărului de operații computaționale necesare în procesul de inserare, căutare și localizare de date reprezentate de funcțiile arborelui `insert()`, `find()`, `locate()`, `remove()`.

Pentru aceasta se consideră un arbore de formă arbitrară ale cărui componente sunt $x_1, x_2, ..., x_n$ luate în ordinea lor crescătoare, reprezentând cazul cel mai defavorabil de introducere a datelor în arborele binar clasic. Datorită ordinii de inserare, un nod x_i va ocupa în prima fază un nod extern unic determinat, în cele din urmă ocupând o poziție pe traseul de căutare al nodului imaginar x_k.

Introducerea datelor în arborele inițial vid se face prin ocuparea unuia dintre nodurile sale externe într-o primă fază. Astfel, x_1 va ocupa singurul nod exterior al arborelui, devenind rădăcina acestuia. Prin urmare, probabilitatea ca nodul x_1 să se situeze pe traseul de căutare al nodului imaginar x_k este 1.0, x_1 fiind părinte al acestuia.

Următoarea componentă introdusă în arbore este x_2 care va ocupa unul dintre cele două noduri externe ale arborelui în configurația sa curentă, probabilitatea ca x_2 să se situeze pe traseul de căutare al lui x_k fiind 0.5.

În general, prin introducerea unei date x_i, $i = 1, ..., k - 1$, probabilitatea ca x_i să ocupe un anumit nod extern este aceeași pentru toate nodurile externe. Deoarece numai într-o singură poziție în arbore, x_i se va situa pe traseul de căutare al lui x_k, probabilitatea ca x_i să se situeze pe traseul de căutare al lui x_k este $1/i$.

Prin urmare, adâncimea probabilă a datei x_k poate fi determinată cu ajutorul expresiei:

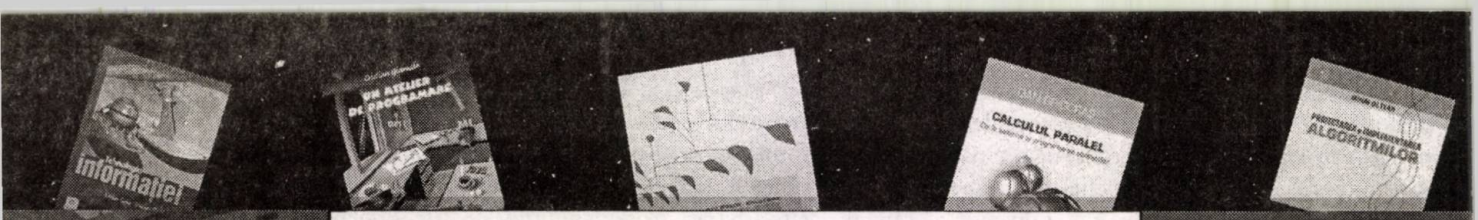
$$d_{x_k}^{\max} = 1 + \sum_{i=1}^{k-1} \left(\frac{1}{i}\right) \equiv O(\log n) \quad k \le n.$$

Deoarece numărul de operații necesare pentru inserare, căutare și localizare este proporțional cu adâncimea în arbore a lui x_k rezultă că aceste procese au loc într-un timp de ordinul $O(\log n)$ pentru acest tip de arbore binar.

Bibliografie

1. S.B. Lipmann, J. Lajoie, *C++ Primer*, Addison-Wesley, Third Ed., 1999.
2. B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1991.
3. K. Mulmuley, *Computational Geometry: An introduction through randomized algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1994.
4. K.E. Gorlen, S.M. Orlow, P.S. Plexico, *Data Abstraction and Object Oriented Programming in C++*, Willey, Chichester, England, 1990.

Domul Adrian Vasilescu este asistent de programare la Universitatea Carleton din Ottawa, Canada. Poate fi contactat prin e-mail la adresa avasiles@mrco2.carleton.ca.

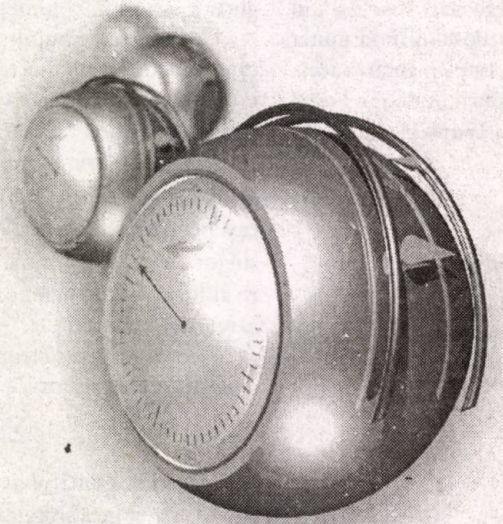


COMPUTER LIBRIS AGORA

DAN GRIGORAȘ

CALCULUL PARALEL

De la sisteme la programarea aplicațiilor



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate, dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Style & Fashion? (3)

Structură și stil în PROGRAMARE

Bazil Pârv

Ultimul episod al serialului nostru conține detalii referitoare la modul în care ar trebui implementați diferiții algoritmi folosiți în rezolvarea unei anumite probleme. Vom exemplifica această modalitate pentru o problemă simplă și anume rezolvarea unei ecuații de gradul I.

În episodul anterior am prezentat implementarea mai multor algoritmi folosiți în programe. Nu am tratat elementele specifice problemei de rezolvat, și anume tipurile de date `TipDate` și `TipRezultate` și subprogramele `Antet`, `CitesteDate`, `CitesteDate2`, `Calculeaza`, `AfiseazaRezultate`.

Rezolvarea ecuației de gradul I

Enunțul acestei probleme este: *să se rezolve ecuația de gradul I:* $a \cdot x + b = 0$.

Specificarea sa este următoarea:

```
Specificarea EcGr1 este:      { varianta 0 }
  Date: a,b ∈ R
  Rezultate: s ∈ R astfel încât a * s + b = 0
SfSpecificare { EcGr1 }
```

Din păcate, specificarea de mai sus nu este completă, deoarece nu cuprinde toate situațiile posibile. Fiindcă nu am impus nici o condiție asupra coeficienților a și b, există două cazuri neacoperite. Discuția completă a ecuației cuprinde următoarele situații:

- $a \neq 0$: ecuația are soluția unică $s = -b / a$;
- $a = 0$ și $b \neq 0$: ecuația este incompatibilă (nu are nici o soluție);
- $a = 0$ și $b = 0$: ecuația este nedeterminată (are o infinitate de soluții).

Pentru a cuprinde toate situațiile de mai sus în specificarea problemei, introducem o variabilă de stare numită `cod`, cu următoarea semnificație:

- `cod = 0` pentru $a \neq 0$: ecuația are soluția unică $s = -b / a$;
- `cod = 1` pentru $a = 0$ și $b \neq 0$: ecuația este incompatibilă (nu are nici o soluție);
- `cod = 2` pentru $a = 0$ și $b = 0$: ecuația este nedeterminată (are o infinitate de soluții).

Cu aceste notații, specificarea de mai sus devine:

```
Specificarea EcGr1 este:      { varianta 1 }
  Date: a,b ∈ R
  Rezultate:
    cod = 0 ºi s ∈ R astfel încât a * s + b = 0
                                  dacã a ≠ 0 sau
    cod = 1 dacã a = 0 ºi b ≠ 0 sau
    cod = 2 dacã a = 0 ºi b = 0 sau
SfSpecificare { EcGr1 }
```

Din această specificare putem extrage componentele tipurilor de date `TipDate` și `TipRezultate` proiectate generic în cadrul episodului anterior:

```
type TipDate = record
       a,b:Real  { coeficientii ecuatiei de }
                 { gradul I: a * x + b = 0 }
     end; { TipDate }

     TipRezultate = record
       cod:Byte; { caracterizeaza solutia   }
         { cod = 0: solutie unica (a <> 0) }
         {                  memorata in s }
         { cod = 1: incompatibilitate      }
```

```
        {                    (a = 0, b <> 0) }
        { cod = 2: nedeterminare             }
        {                    (a = 0, b = 0) }
      s:Real { solutia ecuatiei -            }
             {    numai in cazul cod = 0 }
    end; { TipRezultate }
```

Implementări procedurale

Deoarece acum problema care trebuie rezolvată este specificată complet, putem să descriem și să implementăm subalgoritmii Antet, CitesteDate, CitesteDate2, Calculeaza și AfiseazaRezultate. Prezentăm în continuare codurile sursă respective.

Implementarea subprogramului CitesteDate

```
procedure CitesteDate(var DI:TipDate);
{ citeste datele de intrare ale problemei
  parametru: DI - datele de intrare ale
                              problemei (OUT) }
begin
  Write('a=');
  Readln(DI.a);
  Write('b=');
  Readln(DI.b)
end; { CitesteDate }
```

Implementarea subprogramului CitesteDate2

```
procedure CitesteDate2(var DI:TipDate;
                       var SuntDate:Boolean);
{ citeste datele de intrare ale problemei
  (daca exista)
  parametri: DI - datele de intrare ale
                                problemei (OUT)
             SuntDate - specifica daca s-au
               citit datele de intrare (OUT)
             SuntDate = true - s-au citit
                           datele de intrare
             SuntDate = false - nu s-au citit
                          datele de intrare }
var c:Char;
begin
  repeat
    Write('Exista date de intrare (D/N): ');
    Readln(c)
  until c in ['D','d','N','n'];
  if c in ['D','d'] then
    begin
      SuntDate:=true;
      Write('a=');
      Readln(DI.a);
      Write('b=');
      Readln(DI.b)
    end
  else SuntDate:=false
end; { CitesteDate2 }
```

Implementarea subprogramului Antet

```
procedure Antet;
{ afiseaza antetul problemei de rezolvat }
begin
  Clrscr;
  Write('Programul calculeaza solutia ',
        'ecuatiei de gradul I a * x + b = 0')
end; { Antet }
```

Implementarea subprogramului Calculeaza

```
procedure Calculeaza(DI:TipDate;
                     var RC:TipRezultate);
{ calculeaza rezultatele RC pe baza datelor de
                                  intrare DI
  parametri: DI - datele de intrare ale
                                  problemei (IN)
             RC - rezultatele calculate
                                      (OUT) }
begin
  if DI.a = 0 then
    if DI.b = 0 then
      RC.cod := 2
    else RC.cod := 1
  else
    begin
      RC.cod := 0;
      RC.s := -DI.b / DI.a
    end
end; { Calculeaza }
```

Implementarea subprogramului AfiseazaRezultate

```
procedure AfiseazaRezultate(DI:TipDate;
                            RC:TipRezultate);
{ afiseaza rezultatele si (eventual) datele de
                                    intrare
  parametri: DI - datele de intrare ale
                                  problemei (IN)
             RC - rezultatele calculate (IN) }
begin
  Write('S-a rezolvat ecuatia: ');
  if DI.a <> 0 then
  begin
    Write(DI.a:10:2, '*X ');
    if DI.b > 0 then
      Write('+ ')
  end;
  Write(DI.b:10:2);
```

```
  Writeln(' = 0');
  case RC.cod of
    0: WriteLn('Ecuatia are solutia unica: ',
                                    RC.s:10:2);
    1: WriteLn('Ecuatia este incompatibila ',
                              '(a = 0, b <> 0)');
    2: WriteLn('Ecuatia este nedeterminata ',
                             '(a = 0, b = 0)')
  end
end; { AfiseazaRezultate }
```

Implementări modulare

Dacă studiem cele patru versiuni ale programelor principale, prezentate în cadrul episodului anterior, constatăm că declarațiile tipurilor de date `TipDate` și `TipRezultate`, precum și declarațiile subprogramelor `Antet`, `CitesteDate`, `Calculeaza` și `AfiseazaRezultate` se repetă în fiecare dintre ele (cu excepția versiunii 2, în care `CitesteDate` este înlocuit cu `CitesteDate2`). Este natural, prin urmare, să proiectăm un modul care să conțină toate aceste declarații și apoi să-l folosim în toate versiunile programelor.

Un singur modul

O primă tentativă de modularizare (radicală) va include în modulul `UEcGr1` toate declarațiile din programe, păstrând în acestea numai declarațiile de variabile globale și corpul programului principal. Interfața modulului `UEcGr1` este:

```
Unit UEcGr1;
          { rezolvarea ecuatiei de gradul I }
interface
  type { declaratiile tipurilor de date      }
       {                TipDate si TipRezultate }
    TipDate = record
      a,b:Real  { coeficientii ecuatiei de }
                { gradul I: a * x + b = 0 }
    end; { TipDate }
    TipRezultate = record
      cod:Byte; { caracterizeaza solutia   }
          { cod = 0: solutie unica (a <> 0) }
          {                   memorata in s }
          { cod = 1: incompatibilitate      }
          {                  (a = 0, b <> 0) }
          { cod = 2: nedeterminare          }
          {                  (a = 0, b = 0) }
      s:Real { solutia ecuatiei -           }
             {      numai in cazul cod = 0 }
    end; { TipRezultate }
    { declaratia tipului de date TipOperatie }
    TipOperatie = (Terminare, Citire, Calcul,
                                    Afisare);
    { declaratiile subprogramelor publice }
  procedure Antet;
  procedure CitesteDate(var DI:TipDate);
  procedure CitesteDate2(var DI:TipDate;
                         var SuntDate:Boolean);
  procedure Calculeaza(DI:TipDate;
                       var RC:TipRezultate);
  procedure AfiseazaRezultate(DI:TipDate;
                              RC:TipRezultate);
  function Continuare:Boolean;
  procedure AfiseazaMeniu(var op:TipOperatie);
  procedure Mesaj(m:string);
```

Partea de declarații din programele prezentate în episodul anterior se va înlocui cu instrucțiunea:

```
uses UEcGr1;
```

Am obținut astfel programe generale, în care tot ce este specific problemei de rezolvat este "ascuns" într-un modul. Spre exemplu, varianta 3 are forma:

```
Program V3;
{ executie cu seturi repetate de date  }
{ continuarea sau terminarea executiei este }
{ separata de citirea datelor de intrare  }
uses UEcGr1;
          { rezolvarea ecuatiei de gradul I }
{ declaratia variabilelor globale }
var D:TipDate; { datele de intrare }
    R:TipRezultate; { rezultatele }
Begin
  repeat
    Antet; { subprogram de prezentare }
    CitesteDate(D);
          { subprogram care citeste datele }
    Calculeaza(D,R);
        { subprogram care face prelucrarile }
    AfiseazaRezultate(D,R)
     { subprogram care afiseaza rezultatele }
  until not Continuare
End. { V3 }
```

Înlocuind în fraza **uses** numele modulului `UEcGr1` cu numele altui modul (în care este implementată rezolvarea altei probleme) programul va funcționa normal. Am obținut astfel patru variante generale de programe, în care am izolat particularitățile problemei de rezolvat într-un modul.

Două module

Dacă analizăm interfața modulului `UEcGr1`, constatăm că ea conține și declarații publice, care nu sunt specifice problemei de rezolvat: tipul de date `TipOperatie` (necesar pentru procedura `AfiseazaMeniu`) și subprogramele Con-

tinuare, AfiseazaMeniu și Mesaj.

Acestea sunt aceleași pentru orice problemă și, prin urmare, este natural să le punem în alt modul, pe care îl vom numi UService.

Prin urmare, am "spart" modulul UEcGr1 în două module mai mici: UService (care este general) și UEcGr1V2 (care va conține numai elementele specifice rezolvării problemei, adică tipurile de date TipDate și TipRezultate și subprogramele Antet, CitesteDate, Calculeaza și AfiseazaRezultate.

Interfața modulului UService este:

```
Unit UService;
interface
   { declaratia tipului de date TipOperatie }
  type TipOperatie = (Terminare, Citire,
                      Calcul, Afisare);

     { declaratiile subprogramelor publice }
  function Continuare:Boolean;
  procedure AfiseazaMeniu(var op:
                          TipOperatie);
  procedure Mesaj(m:string);
```

În interfața modulului UEcGr1V2 vor rămâne numai declarațiile.

```
Unit UEcGr1;
        { rezolvarea ecuatiei de gradul I }
interface
  type { declaratiile tipurilor de date     }
       {          TipDate si TipRezultate }
    TipDate = record
      a,b:Real  { coeficientii ecuatiei de }
                { gradul I: a * x + b = 0 }
    end; { TipDate }
    TipRezultate = record
      cod:Byte; { caracterizeaza solutia   }
          { cod = 0: solutie unica (a <> 0) }
          {                  memorata in s }
          { cod = 1: incompatibilitate     }
          {                (a = 0, b <> 0) }
          { cod = 2: nedeterminare         }
          {                (a = 0, b = 0) }
       s:Real { solutia ecuatiei -         }
              {    numai in cazul cod = 0 }
    end; { TipRezultate }

     { declaratiile subprogramelor publice }
  procedure Antet;
  procedure CitesteDate(var DI:TipDate);
  procedure CitesteDate2(var DI:TipDate;
                         var SuntDate:Boolean);
  procedure Calculeaza(DI:TipDate;
                       var RC: TipRezultate);
  procedure AfiseazaRezultate(DI:TipDate;
                              RC:TipRezultate);
```

Părțile de declarații din programe se vor înlocui cu instrucțiunea **uses** UEcGr1V2; (pentru primele două versiuni), respectiv **uses** UEcGr1V2,UService; (pentru celelalte două versiuni).

Noua arhitectură obținută izolează și mai bine specificul problemei de rezolvat de programele generale obținute. Pe lângă scheletul acestora (în care singurul parametru este numele modulului care conține rezolvarea problemei), am obținut și un modul general UService, care conține proceduri de serviciu, necesare pentru ultimele două versiuni ale programelor.

Concluzii

Oare de câte ori au rostit elevi și studenți: "*De ce nu e bine? Doar funcționează!*"?

Da, stimați cititori, este nevoie de încă ceva: programele nu se scriu *oricum* - se proiectează astfel încât oricând ar dori autorul programului respectiv, sau un alt programator să-l "priceapă" repede, să poată refolosi părți din el, să-l poată folosi în condiții diferite, operând un număr cât mai mic de modificări.

Dacă un program nu este scris respectând paradigma programării modulare, nu există "părți", dacă nu se ține cont de necesitatea scrierii unor programe flexibile și parametrizate, reutilizarea poate să devină imposibilă iar orice modificare va însemna, de fapt, scrierea unui program complet nou.

În aceste condiții etapele care trebuie respectate în realizarea unui program în condiții de dezvoltare incrementală vor fi:

1. Scrierea unor programe ținând cont de cerința de funcționalitate. Mai întâi se vor proiecta algoritmi (subalgoritmi) cât mai simpli; se va pune accent pe structura programului. Funcționalitatea nu poate fi realizată decât dacă se respectă cerințele impuse de problema de rezolvat.
2. Dacă prima versiune a algoritmilor proiectați nu se dovedesc a fi suficient de performanți, respectiv, permit optimizări, acestea se vor căuta și se vor implementa.
3. Nu pot fi ignorate acele modificări necesare de "înfrumusețare" care asigură o utilizare ușoară chiar și pentru un "beneficiar neinstruit" printr-o interfață prietenoasă, realizată și aceasta cât se poate de simplu.

Dl. prof. univ. Bazil Pârv este cadru didactic la Universitatea "Babeș-Bolyai" din Cluj. Poate fi contactat prin e-mail la **bparv@cs.ubbcluj.ro**.

SISTEME nestandard. METODE noi de rezolvare

Laviniu Bejenaru

Modelul uman de viață a făcut ca permanet să ne lovim de rezolvarea unor probleme. A rezolva o problemă înseamnă a pleca de la anumite date cunoscute și a încerca să găsim soluții potrivite. Organizarea datelor și a relațiilor dintre ele atât cunoscute cât și necunoscute este realizată sub forma a ceea ce numim ecuație. Ecuațiile au, în afară de aspectul strict matematic al unor relații reci, o anumită semnificație. Dacă problemele necesită mai multe relații, atunci aceste relații, deși independente, se pot grupa în sisteme de ecuații.

Cu ajutorul sistemelor de ecuații se pot rezolva, în principiu, mai multe probleme dacât cu ajutorul ecuațiilor; de fapt, dacă abstractizăm, o ecuație poate fi privită ca un sistem de ecuații în care numărul de ecuații componente este egal cu unu. Posibilitatea rezolvării sistemelor conduce la rezolvarea problemelor și chiar la o interpretare a lor.

Prezentare

Problemele cărora le căutăm rezolvare conduc adesea la ecuații și sisteme de ecuații care nu se încadrează într-un anumit tip, cum ar fi ecuații de gradul unu, doi, binome etc., sisteme de ecuații liniare sau sisteme simetrice.

Putem vorbi astfel despre ecuații nestandard sau despre sisteme de ecuații nestandard sau, pe scurt, sisteme nestandard.

Desigur, problemele complexe din industrie sau economie generează sisteme a căror tratare nu se atinge în liceu, cum ar fi sistemele de ecuații diferențiale. Totuși, în cele ce urmează, vom trata aici doar cazul unor sisteme elementare, nestandard, care se studiază și în învățământul preuniversitar.

Să dăm un exemplu:

$$(1)\quad \begin{cases} x(x+y+4z)-4z^2=0 \\ y(x+y+4z)-8z^2=0 \\ xyz=8 \end{cases}$$

Rezolvarea sistemelor de acest tip poate fi destul de dificilă. Găsirea anumitor amănunte sau observații poate conduce de cele mai multe ori la rezolvare.

Prin intermediul programului *MAPLE V*, însă, avem la dispoziție mai multe instrumente pentru a rezolva aceste sisteme.

Preambul

Pentru a putea folosi funcții și proceduri dintr-un anumit *unit*, în *Pascal* trebuie să informăm compilatorul prin instrucțiunea **uses** `NumeUnit;` (de exemplu, **uses** `Crt,Graph;`).

În *C/C++* același lucru este realizat prin indicarea numelui bibliotecii *header*. Acest lucru se realizează prin `#include <NumeBiblioteca>` (de exemplu, `#include <stdio.h>`).

În *Maple V* pentru a folosi funcțiile dintr-o anumită bibliotecă va trebui să solicităm includerea bibliotecii respective. Această operație se realizează prin:

```
> with(NumeBiblioteca);        sau
> with(NumeBiblioteca):
```

Sintaxa obligă terminarea acestei comenzi cu unul dintre caracterele `";"` sau `":"`.

Noi vom folosi biblioteca matematică `linalg`, deci vom avea un element de forma:

`> with(linalg);` pentru a afișa funcțiile conținute

sau

`> with(linalg):` pentru a inhiba afișarea funcțiilor conținute

Observație

Mediul *MAPLE V* este *case-sensitive*, exact ca și *C++* sau *Java*, adică se face diferență între literele mari și cele mici.

Din acest motiv trebuie să fim atenți la scrierea identificatorilor pentru biblioteci și funcții sau proceduri predefinite.

În general, după fiecare acțiune adresată sistemului și terminată prin ";" are loc efectuarea ei cu ecou, adică este afișat pe centrul ecranului rezultatul, chiar dacă facem și o simplă atribuire prin ":=". Dacă terminăm comanda prin ":", ea este efectuată fără ecou, adică nu va fi afișat rezultatul ei.

Ecuațiile pot fi indicate sistemului prin instrucțiuni de atribuire sau pot fi scrise direct în funcția de rezolvare a sistemului. Preferăm scrierea ecuațiilor separat, prin atribuiri de forma:

```
> ecI:=descriere_ecuatie;
```

De exemplu, pentru prima ecuație a sistemului, avem

```
> ec1:=x*(x+y+4*z)-4*z^2=0;
```

Precizăm faptul că simbolul "^" înseamnă "*ridicat la puterea*". Astfel pentru a^m vom scrie "a^m".

Așadar, x^2 înseamnă x^2 și datorită faptului că operatorul de ridicare la putere are prioritate mai mare decât operatorul de înmulțire, nu sunt necesare paranteze.

Utilizăm apoi funcția solve pentru rezolvarea efectivă a sistemului.

Liste

Pentru afișarea eficientă a soluțiilor și selectarea celor reale este util să evidențiem toate soluțiile ecuației prezentate sub forma unei liste.

Lista este un mod de a enumera o serie de elemente într-o ordine bine precizată. În *Maple V* lista este creată prin încadrarea între paranteze drepte sau acolade a unor elemente sau a unei funcții generatoare, formule etc. ce poate să furnizeze mai multe elemente. Forma listei este [secv] sau {secv} unde secv este o secvență de expresii.

Elementele listei sunt separate prin virgulă dacă lista este creată manual. Diferența dintre cele două forme este aceea că, în cazul parantezelor drepte, lista poate conține și elemente identice pe diverse poziții, în timp ce la cea cu acolade toate elementele păstrate în listă după crearea ei sunt distincte.

Vom prezenta mai jos acestă diferență prin crearea, în ambele moduri, a unei liste care conține elemente identice.

```
> {x,y,y};
                        {x, y}
> {y,x,y};
                        {x, y}
> [x,y,y];
                      [x, y, y]
> [y,x,y];
                      [y, x, y]
```

O listă poate fi creată și printr-o enumerare, caz în care se poate folosi tipul subdomeniu $l_i \ldots l_f$, unde l_i este limita inferioară iar l_f este limita superioară.

Va trebui să folosim și funcția seq pentru a crea efectiv enumerarea.

Sintaxa acestei funcții este:

```
seq(exp, ind = li .. lf);
```

unde exp este o expresie iar ind este un *identificator de indice* care parcurge domeniul dintre limita inferioară l_i și limita superioară l_f. Atât limita inferioară cât și cea superioară sunt valori numerice. Iată un exemplu în care prezentăm atât acțiunea cât și efectul.

```
>L := [seq(x[i],i=1..4)];
        L := [x[1], x[2], x[3], x[4]]
```

Vom da acum un exemplu pentru crearea unei liste care va conține pătratele numerelor cuprinse între 10 și 25.

```
>L1:=[seq(i^2,i=10 .. 25)];
L1:=[100, 121, 144, 169, 196, 225, 256, 289,
324, 361, 400, 441, 484, 529, 576, 625, 676,
                729, 784, 841, 900]
```

În articolul *Rezolvarea unei ecuații cu coeficienți numere întregi foarte mari* din numărul 11/1 (ianuarie 2001) al *GInfo* am precizat faptul că *Maple V* este astfel conceput încât să poată lucra cu numere mari. Iată un exemplu în care putem combina o listă în care reținem pătratele unor numere foarte mari cum ar fi numerele cuprinse între 1.900.047.000 și 1.900.047.005:

```
>L1:=[seq(i^2,i=1900047461..1900047465)];
          L1:=[ 3610180354052546521,
  3610180357852641444, 3610180361652736369,
  3610180365452831296, 3610180369252926225 ]
```

Putem afla la fel de ușor lista valorilor radicalilor de orice ordin pentru numere aflate între anumite limite. Vom prezenta lista radicalilor de ordinul 3 pentru numerele cuprinse între 2 și 10.

Pentru afișarea valorilor efective trebuie să folosim și funcția:

```
evalf(exp, nrc);
```

unde exp este expresia care se evaluează iar nrc este numărul de cifre folosit pentru afișarea valorilor.

Pentru a calcula radicalul de ordinul n al unui număr a vom scrie a^(1/n). Deci, radicalul de ordinul 3 al valorii unei variabile i care parcurge domeniul se va obține cu ajutorul construcției i^(1/3); valorile propriu-zise se determină prin evaluare, deci vom scrie evalf(i^(1/3),7). În acest caz am indicat faptul că dorim o afișare cu 7 cifre zecimale.

```
>L2_0=[seq(i^(1/3),i=2..10)];
 L2_0=[ 21/3, 31/3, 41/3, 51/3, 61/3, 71/3,
          81/3, 91/3, 01/3 ]
>L2:=[seq(evalf(i^(1/3),7),i=2..10)];
 L2:=[1.259921, 1.442250, 1.587401, 1.709976,
   1.817121, 1.912931, 2.000000, 2.080084,
                2.154435]
```

Listele în care sunt folosite acoladele sunt considerate mulțimi. Când vorbim despre mulțimi ne punem imediat problema operațiilor care pot fi efectuate asupra lor.

Asupra listelor mulțime putem efectua operațiile uzuale de *reuniune*, *intersecție* și *diferență*. Diferența simetrică apare ca o îmbinare a diferenței și reuniunii, cu alte cuvinte, a operațiilor elementare precizate mai sus.

Reuniunea

Pentru reuniune avem funcția union. Sintaxa ei este:

```
`union` (m1,m2,...,mp);
```

unde m1, m2, ..., mp sunt mulțimile care se vor reuni.

Ca exemplu, avem:

```
>`union` ({3,4,67,33,5,89},{5,8,33,21,77},
{3,77,6});
    {8, 3, 4, 5, 6, 21, 33, 67, 77, 89}
```

Observație

Cuvântul cheie union trebuie să fie încadrat de caracterul "`" atât în stânga cât și în dreapta. Acest caracter nu este apostroful normal, ci este caracterul care are codul ASCII 96.

Se poate vedea că union este un operator *n*-ar, adică îl putem aplica simultan asupra a *n* operanzi.

În cazul în care folosim doar doi operanzi, forma operatorului poate fi și A union B unde A și B sunt cele două mulțimi care se reunesc. În acest caz union nu mai este încadrat de caracterele "`". Iată și un exemplu:

```
> {3,4,5,6} union {3,6,9,12};
                {3,9,12,4,5,6}
```

Intersecția

Pentru intersecție avem funcția intersect. Sintaxa ei este:

```
`intersect` (m1,m2,...,mp);
```

unde m1, m2, ..., mp sunt mulțimile care se vor intersecta.

Ca exemplu, avem:

```
>`intersect` ({3,4,7,33},{4,6,7,10,21,33},
{4,28,33});
                {4, 33}
```

Observația de mai sus este valabilă și pentru funcția intersect.

Se poate vedea că operatorul intersect este și el unul *n*-ar, adică îl putem aplica simultan asupra a *n* operanzi.

În cazul în care folosim doar doi operanzi, forma operatorului poate fi și A intersect B unde A și B sunt cele două mulțimi care se intersectează. Iată și un exemplu:

```
> {3,4,5,6} intersect {3,6,9,12};
                {3,6}
```

Diferența

Pentru diferență avem funcția minus. Sintaxa ei este

```
`minus` (m1,m2);
```

unde m1 și m2 sunt mulțimile pentru care se va efectua diferența.

Ca exemplu, avem:

```
>`minus` ({3,4,7,33},{4,6,7,21});
                {3, 33}
```

Se poate vedea că operatorul minus nu este unul *n*-ar, adică nu poate fi aplicat asupra a *n* operanzi.

Forma acestui operator poate fi și A minus B unde A și B sunt cele două mulțimi pentru care se face diferența. Iată și un exemplu:

```
> {3,4,5,6} minus {3,6,9,12};
                {4,5}
```

Diferența simetrică

Această operație se realizează prin intermediul diferenței și reuniunii, deoarece $A \Delta B = (A \setminus B) \cup (B \setminus A)$.

Astfel, vom combina funcțiile minus și union. Un exemplu este:

```
>a:={3,4,5,6};
                a:={3,4,5,6}
>b:={3,6,9,12};
                b:={3,6,9,12}
>dif_1:=a minus b;
                dif_1:={4,5}
>dif_2:=b minus a;
                dif_2:={9,12}
>dif_sim:=dif_1 union dif_2;
            dif_sim:={9,12,4,5}
```

Numere reale și complexe

Mediul *Maple V* acceptă lucrul atât cu numere reale cât și cu numere complexe în forma a+b*I. Se știe că un număr real poate fi privit ca un număr complex cu partea imaginară nulă.

Observație

Partea imaginară este indicată cu ajutorul caracterului "I", adică prin literă mare pentru a avea efect în calcule.

Suma, diferența sau produsul a două sau mai multe numere complexe poate fi realizată natural, prin operatorii corespunzători. Avem "+" pentru adunare, "-" pentru scădere și "*" pentru înmulțire. Trebuie menționat faptul că atunci când scriem un număr complex, între partea imaginară și simbolul "I", trebuie să apară și operatorul de înmulțire "*". Iată și un exemplu:

```
>(2+6*I) + (3-4*I);
                5 + 2I
>(2+6*I) - (3-4*I);
                -1 + 10I
>(2+6*I)*(3+4*I)*(6+7*I);
                -290 + 30I
```

Există operații specifice care pot fi efectuate asupra numerelor complexe. Astfel, putem extrage partea reală și partea imaginară a unei expresii de numere complexe prin utilizarea funcțiilor Re, respectiv Im. Sintaxa acestora este Re(exp), respectiv Im(exp), unde exp este o expresie. Iată și exemplele corespunzătoare:

```
>Re( (2+6*I)*(3+4*I)*(6+7*I) );
                -290
>Im( (2+6*I)*(3+4*I)*(6+7*I) );
                30
```

Tot pentru numerele complexe există funcția evalc. Aceasta realizează o evaluare simbolică a unei expresii, forțând scrierea în forma standard a+b*I. Sintaxa ei este evalc(exp), unde exp este o expresie. Un exemplu este prezentat în continuare:

```
> 3^(4+2*I);
                        3(4+2I)
> evalc( 3^(4+2*I) );
     81cos(2 ln(3)) + 81 I sin(2 ln(3))
```

Instrucțiuni

Puterea mediului *Maple V* nu constă doar din numeroase funcții matematice predefinite. Putem defini propriile funcții și putem profita din plin de faptul că *Maple V* este special realizat astfel încât să lucreze cu numere mari. Dar nici acestea nu ar fi suficiente; este nevoie, ca în orice limbaj, de *instrucțiuni*.

Mediul *Maple V* pune la dispoziție instrucțiunea de selecție **if - fi**, precum și instrucțiunea repetitivă **for - while - do - od**. Aceste instrucțiuni simulează instrucțiunile tradiționale `if` și `for`, însă le și adaptează. Trebuie să remarcăm de la bun început că fiecare dintre instrucțiuni are și o secvență de terminare a sa, exact ca în limbajul *FoxPro*.

Instrucțiunea if - fi

Sintaxa acestei instrucțiuni este:

```
if exp_cond
  then instr
| elif exp_cond
    then instr  |
    | else  instr  |
fi
```

unde `exp_cond` este o expresie condițională, iar `instr` este o secvență de instrucțiuni.

Construcția | | indică faptul că elementul conținut în interior este opțional. Construcția **elif** este folosită pentru a permite o nouă selecție, în măsura în care cazul anterior nu a fost sufiecient de semnificativ.

O mențiune importantă este aceea că prezența elementului **elif** poate fi repetată ori de câte ori este nevoie. Putem realiza astfel și o selecție multiplă mai eficientă decât o serie de instrucțiuni **if** tradiționale imbricate.

Instrucțiunea for - while - do - od

Sintaxa acestei instrucțiuni este:

```
| for num | | from expr_i | | by expr_inc |
                | to expr_f | | while expr_c |
  do
    instr
  od;
                        sau
| for num | | in expr_i | | while expr_c |
  do
    instr
  od;
```

- `num` este numele variabilei care controlează ciclul;
- `expr_i` este expresia folosită pentru inițializarea variabilei `num`;
- `expr_inc` este expresia folosită pentru incrementarea variabilei `num` la fiecare executare a corpului ciclului;
- `expr_f` este expresia folosită pentru a indica valoarea finală la care trebuie să ajungă variabila `num`;
- `expr_c` este o expresie condițională în urma evaluării căreia se decide dacă trebuie continuat ciclul;
- `instr` este o secvență de instrucțiuni.

Noutatea introdusă și acceptată de *Maple V* constă în faptul că, pe de o parte, se pot face salturi dinamice ale valorilor variabilei în timpul executării ciclului, iar, pe de altă parte, în *Maple V* se creează o legătură mai strânsă între instrucțiunile **for** și **while** tradiționale, îmbinându-le într-o instrucțiune mai puternică.

Prin salturi dinamice ale valorii variabilei în timpul rulării ciclului se înțelege posibilitatea modificării valorii prin care se incrementează de fiecare dată variabila.

În limbajul *Pascal* trecerea variabilei de ciclare la următoarea valoare se făcea strict cu o unitate corespunzătoare tipului ordinal al contorului, dacă nu era precizată explicit o altă incrementare în corpul instrucțiunii **for**. O astfel de incrementare nici nu este recomandabilă.

În limbajul *FoxPro* se admite o incrementare cu un pas oarecare (fix) prin folosirea clauzei **STEP** a instrucțiunii **for - endfor**, dar nu sunt permise salturi dinamice.

În *C/C++* este posibilă incrementarea cu o valoare oarecare și se admit și salturi dinamice.

Vom da un exemplu de program *C++* care realizează salturi dinamice ale valorii de incrementare, lucru care se vede prin afișarea valorilor.

```
# include <stdio.h>
# include <conio.h>

void main(void){
  int i;
  printf("\n");
  for(i=1;i<40;i=2*(i+2))
    printf("%d",i);
  getche();
}
```

Valorile afișate de acest program sunt: `1 6 16 36`. Se observă cum doar prin antetul instrucțiunii **for** se pot modifica în mod dinamic valoarea variabilei `i` care realizează incrementarea. Saltul între două valori este la început `5`, apoi `10` și apoi `20`.

În *Maple V* salturile dinamice sunt realizate prin adăugarea clauzei **by** `expr_inc`. Iată un exemplu:

```
> summ:=0;
> for i from 11 by 2 while i<20 do
>    summ:=summ+I
> od;
                        summ:=0
                       summ:=11
                       summ:=24
                       summ:=39
                       summ:=56
                       summ:=75
```

Se observă că saltul este inițial 13, apoi 15, apoi 17 și apoi 19; deci el este dinamic, și este incrementat cu valoarea lui exp_inc.

Prin acest exemplu, am prezentat o instrucțiune **for - while - do - od** în care sunt prezente atât clauza **by** cât și clauza **while**.

Prin această instrucțiune putem simula atât instrucțiunea **while** tradițională cât și instrucțiunea **for**, prezente în multe limbaje de programare. Acest lucru îl vom folosi în continuare în rezolvarea problemei noastre.

Rezolvarea sistemelor

Vom prezenta acum soluția problemei rezolvării sistemelor prin îmbinarea elementelor prezentate anterior. Nu vom trata strict această problemă, ci metoda de a rezolva astfel de probleme.

Trebuie să mai precizăm că vom prezenta și fișierul **.ms** însoțit de comentariile de rigoare. Un comentariu în *Maple V* se realizează prin prezența caracterului "#" (diez). Comentariul se întinde pe toată linia pe care este prezent caracterul "#".

De asemenea, precizăm faptul că secvențele de instrucțiuni dintr-un fișier pot fi executate secvențial alegând din meniul *Format* opțiunea *Execute Worksheet*. Iată fișierul cu soluția, însoțit de comentariile de rigoare și de răspunsul sistemului *Maple V*:

```
>#  Rezolvarea sistemului (1)

> with(linalg):

> # Ecua iile sistemului:

> ecc1:=x*(x+y+4*z)-4*z^2=0;
            ecc1:=x(x+y+4z)-4z2=0
> ecc2:=y*(x+y+4*z)-8*z^2=0;
            ecc2:=y(x+y+4z)-8z2=0
> ecc3:=x*y*z=8;
                  ecc3:=xyz=8

> # Solu ia final  (se caut  valori reale)

> sol_final:={ };
                sol_final:={ }

> # Scriem solu ia sistemului sub form  de
> # list  pentru a o putea prelucra pe
> # componente

> sol_vect:=[solve({ecc1,ecc2,ecc3},{x,y,z})];
 sol_vect:=[ { z=RootOf(_Z3-9), y= , x= =},
   {y=-4, z=1, x=-2}, {z=RootOf(_Z3+_Z+1),
             y=-4RootOf(_Z3+_Z+1),
          x=-2 RootOf(_Z3+_Z+1)} ]

> # Caut solu ia real

> sol_vect[2];
                {y=-4, z=1, x=-2}

> # Solu iile reale le adaug solu iei finale

> sol_final:=sol_final union {sol_vect[2]};
     sol_final:={ {y=-4, z=1, x=-2} }

> # Tratez cazul generat prin RootOf (aici am
> # dou  situa ii). Mai pot ad uga r d cini la
> # solu ia final

> # prima situa ie

> zz1:=[ allvalues(RootOf(z^3-9)) ];
             zz1:=[ 91/3,  ,   ]

> # Tratez un caz mai general; pot avea
> # diverse dimensiuni.
> # Aleg dimensiunea

> dimm:=3;

> # Calculez solu iile primului caz

> for i from 1 to dimm do
>   sol_exact[i]={ (2/3)*zz1[i],
    (4/3)*zz1[i],  zz1[i] }
> od;
          sol_exact1={ 91/3 ,   ,  }
            sol_exact2={  ,   ,  }
            sol_exact3={  ,   ,  }

> #Folosesc estim ri exacte ale solu iilor

> nr_cifre:=16;
> for i from 1 to dimm do
>   sol_apr[i]:={evalf(Re(evalc((2/3)*
    zz1[i])), nr_cifre) + I *
    evalf(Im(evalc((2/3)*zz1[i])), nr_cifre),
    evalf(Re(evalc((4/3)*zz1[i])), nr_cifre)
    + I * evalf(Im(evalc((4/3)*zz1[i])),
    nr_cifre), evalf(Re(evalc(zz1[i])),
    nr_cifre) + I * evalf(Im(evalc(zz1[i])),
    nr_cifre) }
> od;
       sol_apr1:={ 1.386722548701269,
  2.080083823051904, 2.773445097402538 }
       sol_apr2:={ -.6933612743506346
 +1.200936955176002 I, -1.386722548701269
+2.401873910352005 I , -1.040041911525952
          +1.801405432764004 I}
     sol_apr3:={ -.6933612743506346 -
 1.200936955176002 I, -1.386722548701269
- 2.401873910352005 I, -1.040041911525952
          -1.801405432764004 I }
```

```
> # adaug la solu ia final^a doar solu ia real^a

> sol_final:=sol_final union { sol_aprox[1] };
      sol_final:={ {z=1, y=-4, x=2}, {
    1.386722548701269, 2.080083823051904,
          2.773445097402538 } }

>#  Tratez cazul generat prin RootOf (am dou^a
>#  situa ii). Mai pot ad^auga r^ad^acini la
>#  solu ia final^a

>#  a doua situa ie

> zz2:=[allvalues(RootOf(z^2+z+1))];
                zz2:=[   ,   ]

> # Tratez un caz mai general ; pot avea
> # diverse dimensiuni.
> # Aleg dimensiunea

> dimm:=2;

> # Calculez solu iile cazului al doilea

> for i from 1 to dimm do
>     sol_exact[i]={ (2/3)*zz2[i],
      (4/3)*zz2[i], zz2[i] }
> od;
            sol_exact1:={   ,   }
          sol_exact2:={{   ,   }}
```

```
> # Observ c^a nu am nici o solu ie real^a
> # Adaug la solu ia final^a doar solu ia real^a
> # (este cea vid^a)

> sol_final:=sol_final union { };
      sol_final:={ {z=1, y=-4, x=2}, {
    1.386722548701269, 2.080083823051904,
          2.773445097402538 } }

> # SOLU IA FINAL^:

> sol_final;
  { {z=1, y=-4, x=2}, { 1.386722548701269,
   2.080083823051904, 2.773445097402538 } }
```

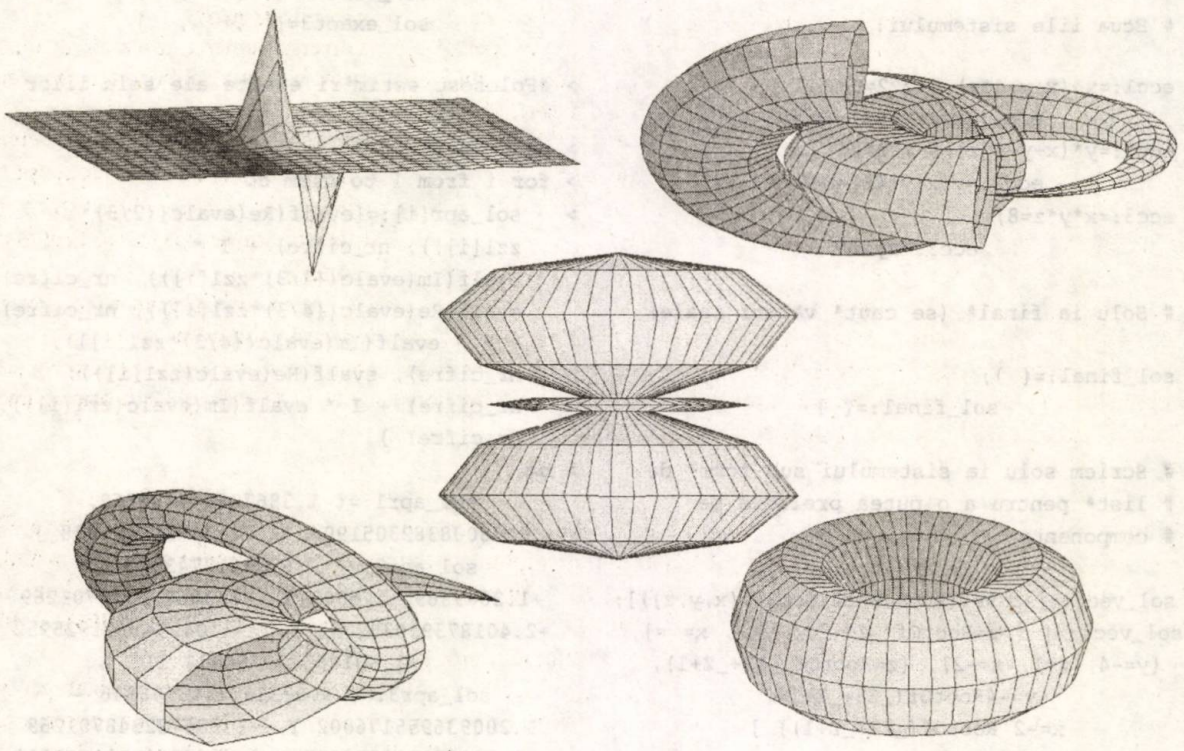
Soluția obținută este dată în forma `{z,y,x}`. Ea are valori exacte sau o aproximație de 15 zecimale (și încă se mai pot adăuga), adică suficient de bună.

Putem spune că legătura dintre un limbaj puternic calculatoriu (dar totuși limbaj) și problemele sistemice este foarte eficientă.

Bibliografie

1. **http://www.maplesoft.com**
2. Ion Nanu, Lucian Tuțescu, *Ecuații nestandard*, Editura Appolo, Craiova, 1994

Dl. prof. drd. Laviniu Bejenaru este cadru didactic la Colegiul Național "Traian" din Drobeta Turnu Severin. Poate fi contactat prin e-mail la adresa **bejenaru@traian.expert.ro**.



Imagini obținute cu MAPLE V

Conjectura PALINDROAMELOR

Tiberiu Socaciu

În numărul 8/4-5 (vara 1998) al revistei Ginfo am publicat un material legat de aritmetica numerelor (foarte) mari (vezi [Socaciu98]) în care am amintit de Conjectura palindroamelor. În articol am precizat că enunțul rămâne încă o conjectură pentru baza 10, falsitatea conjecturii fiind demonstrată pentru cazul sistemului de numerație binar.

Recapitulare

Vom recapitula acum câteva detalii referitoare la noțiunea de *palindrom*.

Definiție

Pentru o secvență de simboluri $w = a_1a_2...a_n$, definim funcția oglindă ca fiind oglindă(w) = $a_na_{n-1}...a_1$. Spunem că secvența w este o secvență palindromică sau că w este un palindrom dacă w = oglindă(w). Spunem că un număr X dat prin reprezentarea sa în baza B este un palindrom dacă reprezentarea lui X în baza B este o secvență palindromică.

Exemplu

Următoarele numere sunt palindroame în orice sistem de numerație cu baza mai mare sau egală cu 10: `1912191`, `9009`.

Unii matematicieni au fost curioși și au început să se "joace" cu numerele; ei au obținut tot felul de rezultate interesante. Pentru un număr oarecare *X* dat în baza *B*, definim următorul proces iterativ: *se adună la primul pas numărului X oglinditul său, repetând succesiv procedeul cu rezultatul obținut.*

Exemplu

Vom exemplifica acest proces pentru numărul 87 în baza 10. La primul pas adunăm numărul *X* cu oglinditul său; la următorul pas adunăm la rezultatul obținut oglinditul acestuia ș.a.m.d.

- *pasul* 1: `87 + 78 = 165`
- *pasul* 2: `165 + 561 = 726`
- *pasul* 3: `726 + 627 = 1353`
- *pasul* 4: `1353 + 3531 = 4884`

Ultimul rezultat este un palindrom care s-a obținut după patru pași.

Enunțul conjecturii

Observăm că după un număr finit de pași am ajuns la un palindrom.

Întrebarea pe care ne-o punem este următoarea: *pornind cu un număr oarecare se ajunge întotdeauna la un palindrom după un număr finit de pași?*

Verificând pentru diverse numere, observăm că unele numere ajung mai repede la forma palindromică, în timp ce altele mult mai greu.

Se pare, totuși, că acest enunț este adevărat indiferent de numărul cu care se pornește, dar nimeni nu a reușit să demonstreze valoarea de adevăr a acestei conjecturi.

Cel care a enunțat această conjectură este celebrul *Martin Gardner* (vezi [Gardner]).

Pentru un număr *X* notăm cu *h*(*X*) numărul de iterații după care se ajunge la formă palindromică.

Evident, datorită faptului că acest enunț este o conjectură, nu putem afirma că *h*(*X*) există pentru orice număr *X*.

An bun, an rău...

Au apărut tot felul de speculații pe această conjectură. De exemplu, s-a încercat o asociere a anilor cu această metodă. S-a obținut astfel că 1994, 1995, 1997 ar fi fost ani buni, iar 1996 un an rău, deoarece valorile *h*(194), *h*(195), *h*(197) sunt mult mai mici decât *h*(196).

Dacă încercăm să verificăm conjectura chiar pentru valori care reprezintă ani calendaristici, vom observa că, din contră, anul 1997 ar fi fost unul rău (*h*(1997) este mai mare decât 4.000.000).

Totuși, conjectura este falsă?

Observăm că enunțul conjecturii nu depinde de baza sistemului de numerație. Prin cercetări s-a demonstrat că bazele de numerație 2^n nu produc palindroame.

Reprezentarea cu repetiție

Fie un număr $X=a_1a_2...a_n$ în configurația căruia sunt identice w cifre consecutive: $a_k = a_{k+1} = ... = a_{k+w-1}=b$. Introducem convenția de notare $X=a_1a_2...a_{k-1}(w, b)a_{k+w}a_{k+w+1}...a_n$. Trebuie remarcat faptul că această notație nu este unică, astfel:

```
12223=1(3,2)3=12(2,2)3=1(2,2)23.
```

Exemplu de inconsistență a conjecturii în baza 2

Vom demonstra inconsistența conjecturii pentru numere de forma $X(n)$ = 10(n, 1)1101(n, 0)00, unde n este un număr natural nenul.

• *pasul* 1:

```
10(n,1)1101(n,0)00 +
00(n,0)1011(n,1)01
------------------
11(n,0)1000(n,1)01
```

• *pasul* 2:

```
 11(n,0)1000(n,1)01 +
 10(n,1)0001(n,0)11
-------------------
101(n,1)1010(n,0)00
= 10(n,1)11010(n,0)00
```

• *pasul* 3:

```
10(n,1)11010(n,0)00 +
00(n,0)01011(n,1)01
-------------------
11(n,0)00101(n,1)01
= 11(n+1,0)010(n+1,1)01
```

• *pasul* 4:

```
11(n+1,0)010(n+1,1)01 +
10(n+1,1)010(n+1,0)11
---------------------
101(n+1,1)101(n+1,0)00
= 10(n+1,1)1101(n+1,0)00
```

Observații

1. Nici unul din numerele obținute în cei patru pași nu este palindrom.
2. După patru pași se ajunge la numărul $X(n + 1)$. Prin urmare, se deduce concluzia că oricare ar fi n natural nenul (de exemplu n = 1), nu se pot obține palindroame pornind de la $X(n)$.

Exemple analoage în baze exponent ale lui 2

Soluția poate fi extinsă cu succes și la baze exponent de 2, după cum se poate observa tabelul următor (pentru bazele 4, 8, 16 și 32):

Baza	Numărul	Iterații
4	10(n,3)3323(n,0)00	6
8	10(n,7)7767(n,0)00	8
16	10(n,F)FFEF(n,0)00	10
32	10(n,V)VVUV(n,0)00	12

Observație

Cifrele F, E din sistemul de numerație hexazecimal, respectiv cifrele V, U din sistemul de numerație cu baza 32 se construiesc după regulile uzuale de atribuire a simbolurilor cifrelor mai mari decât 9.

Generalizarea exemplului

În baza $B = 2^n$, numărul 10(n,A)AAZA(n,0)00, unde cifra A este A = $(B - 1) = 2^{n-1}$ și cifra Z este Z = $(B - 2) = 2^n - 2$ = A - 1 se ajunge la "perioadă" după $2 \cdot (n + 1)$ iterații. Încercați să demonstrați acest lucru!

Soluții sporadice

Alte soluții se pot întâlni și în alte cazuri, unele baze de numerație (cum ar fi 20) generând soluții cu mai mult de 200 de simboluri în reprezentare. Detalii pot fi găsite în [Seal].

Încheiere

Foarte mulți pasionați încearcă la ora actuală să descopere contraexemple pentru a infirma conjectura în diverse sisteme de numerație. Se pare că pentru baza 10 problema este încă deschisă.

Dacă bătălia pentru $h(196)$ s-a dat cu ajutorul unui program C care rula pe un sistem cu *UNIX* (vezi [Pquest]), la ora actuală se folosesc calculatoare special programate precum și sisteme de calcul simbolic de genul lui *Maple*.

Evident, folosind o aritmetică cu numere mari ca cea din [Socaciu95] aveți șanse destul de mari să vă implementați propriul sistem de infirmare a conjecturii palindroamelor pentru diverse baze de numerație.

Notă

Până în acest moment, pentru baza 10, încă nu s-a infirmat conjectura. Așadar, ***spor la treabă***!

Bibliografie

[Gardner] Martin Gardner, *Ha ha ou la comprehension de la mathematique*, Ed. Belin, Paris.

[Pquest] *Program de demonstrare a conjecturii palindroamelor*: **http://www.pquest.com/pquest.c**.

[Seal] *David Seal's Page*: **http://www.seanet.com/dseal.html**

[Socaciu95] Tiberiu Socaciu, *Calcul Simbolic*, teză de licență, *Universitatea "Babeș-Bolyai" Cluj*, (1995).

[Socaciu98] Tiberiu Socaciu, *Aritmetică cu numere mari*, *GInfo* 8/4-5 (vara 1998).

Dl. Tiberiu Socaciu este licențiat și master al Facultății de Matematică din cadrul Universității "Babeș-Bolyai" Cluj. Poate fi contactat prin e-mail la adresa **infodata@theoffice.net** *sau prin fax la* 064/195015.

Performanțele PROCESOARELOR

Mihai Scorțaru

În acest articol vom încerca să prezentăm performanțele care pot fi obținute de anumite procesoare. Pentru aceasta vom prezenta rezultatele obținute de procesoarele actuale folosind benchmark-ul SYSmark 2000, unul dintre cele mai bune produse de acest tip.

Benchmark-uri

Un *benchmark* este un produs software (sau componentă a unui produs) care este folosit pentru a determina performanțele unui sistem (sau a unei componente a unui sistem).

De obicei, aceste programe efectuează anumite operații și măsoară timpul în care acestea au fost efectuate. Rezultatele sunt comparate cu cele obținute de un ***sistem de referință***, iar sistemul testat primește un punctaj.

Există mai multe astfel de produse, fiecare testând anumite caracteristici ale sistemului.

Unul dintre *benchmark*-urile cele mai folosite pentru a testa performanța generală a unui întreg sistem este *SYSmark 2000*. Există și alte astfel de programe, unul dintre cele mai cunoscute fiind *ZD Winstone*.

Performanțele plăcilor grafice pot fi testate cu ajutorul unei facilități oferite de anumite jocuri cum ar fi *Quake III Arena* sau *Unreal Turnament*. Pentru plăcile grafice este foarte important numărul de *refresh*-uri (reafișări ale imaginii) care pot fi efectuate într-o secundă.

SYSmark 2000

Am ales acest *benchmark* deoarece, se pare, el este cel mai folosit produs de acest tip din lume, iar rezultatele obținute reflectă în cea mai mare măsură realitatea.

SYSmark 2000 este produs de compania *BAPCo* (*Business Applications Performance Corporation*).

Prezentare generală

BAPCo a introdus în 1992 conceptul de testare a performanțelor sistemelor pe baza simulării execuției unor aplicații des folosite de utilizatori.

După dezvoltarea primei versiuni a produsului (*SYSmark 92*), numeroase alte companii au realizat diferite *benchmark*-uri folosind conceptul introdus de *BAPCo*.

Versiunea *SYSmark 2000* folosește noi aplicații disponibile pe piață pentru a testa sistemele. O atenție sporită a fost acordată aplicațiilor care sunt folosite pentru a crea produse care sunt folosite pe *Internet*.

SYSmark 2000 folosește 12 aplicații pentru a evalua și caracteriza performanțele unui anumit sistem.

În timp ce *benchmark*-ul rulează, sistemul primește diferite ***sarcini*** pe care trebuie să le execute. Aceste sarcini au fost în așa fel alese încât să reflecte cât mai corecte modul în care sunt folosite aplicațiile de utilizatorul obișnuit.

SYSmark 2000 este disponibil în variante pentru sistemele de operare *Windows '95*, *Windows '98*, *Windows 2000* și *Windows NT 4.0*. În cadrul acestui articol vom folosi rezultatele obținute, utilizând sistemul de operare *Windows '98 Second Edition*.

Modul de punctare

După rularea *benchmark*-ului, sistemul primește un punctaj pentru fiecare aplicație, un punctaj pentru fiecare categorie de aplicații și un punctaj care caracterizează performanța generală a sistemului.

Pentru a determina punctajul corespunzător unei aplicații, se compară timpul de care a avut nevoie sistemul pentru a îndeplini sarcina cu timpul în care a îndeplinit sarcina un sistem considerat ***sistem etalon***. Un punctaj de 100 de puncte indică faptul că sistemul testat are aceeași performanță cu sistemul etalon, un punctaj de 200 de puncte indică o performanță dublă, iar unul de 300 de puncte o performanță triplă.

Punctajul corespunzător unei categorii este determinat calculând ***media geometrică*** a punctajelor obținute pentru fiecare aplicație din categoria respectivă.

Punctajul general este determinat tot pe baza unei medii geometrice, de data aceasta a punctajelor corespunzătoare fiecărei categorii.

Performanțele unui sistem sunt, în principal, afectate de tipul plăcii de bază, tipul și frecvența procesorului, cantitatea de memorie, tipul plăcii video, rezoluția la care este realizată afișarea, tipul discului hard și sistemul de operare folosit. Alte caracteristici ale sistemului cum ar fi tipul plăcii de sunet, viteza unității CD-ROM, dimensiunea discului hard etc. nu au un efect foarte important asupra performanțelor sistemului.

În *SYSmark 2000*, sistemul etalon este următorul:

- **placa de bază**: *Intel 440BX*;
- **procesor**: *Intel Pentium III*;
- **frecvență**: *450 MHz*;
- **memorie**: *128 MB DIMM*;
- **placă video**: *Diamond Viper V770 Ulta, 32 MB*;
- **rezoluție**: *1024 × 768 × 16*;
- **disc hard**: *IBM DJNA 371800*;
- **sistem de operare**: *Microsoft Windows '98 Second Edition*.

Aplicațiile folosite

SYSmark 2000 folosește sarcini pentru 12 aplicații, fiecare dintre ele constând dintr-un *script* care trimite comenzi aplicației respective. Cele 12 aplicații sunt împărțite în două categorii: ***Office Productivity*** și ***Internet Content Creation***.

Prima categorie este formată din următoarele aplicații: *CorelDRAW 9*, *Microsoft Excel 2000*, *Dragon Systems NaturallySpeaking Preferred 4.0*, *Netscape Communicator 4.61*, *Corel Paradox 9*, *Microsoft PowerPoint 2000* și *Microsoft Word 2000*.

Aplicațiile care formează a doua categorie sunt: *MetaCreations Bryce 4*, *Avid Elastic Reality 3.1*, *Adobe Photoshop 5.5*, *Adobe Premiere 5.1* și *Microsoft Windows Media Encoder 4.0*.

În continuare sunt descrise *script*-urile folosite pentru fiecare dintre aceste aplicații.

CorelDRAW 9

- Se preia un desen abstract și se aplică un anumit *Art Stroke* apoi sunt rulate diferite filtre (*spheroid*, *charcoal* etc.).
- Se creează și se manipulează o scenă construită folosind grafica vectorială.
- Se preia o imagine *raster* și se aplică anumite efecte (*add noise*, *blur* etc.).
- Se creează câteva obiecte 3D asupra cărora sunt efectuate câteva manipulări tridimensionale.
- Câteva dintre imaginile create anterior sunt "publicate" sub forma unui *site Web*.

Corel Paradox 9

- Se importă un fișier text de dimensiuni mari și se creează un tabel care conține câteva zeci de mii de intrări.
- Sunt efectuate câteva interogori de tip *SQL* și apoi sunt căutate intrările duplicate.
- Se mai preiau câteva fișiere text care sunt formate și apoi exportate în format *HTML*.
- Sunt deschise câteva formulare de interogare, mai sunt introduse câteva date și apoi sunt generate rapoarte pe baza interogărilor din formulare. Fiecare raport este exportat în format *HTML*.

Microsoft Word 2000

Scriptul pentru *Microsoft Word 2000* efectuează numeroase operații de procesare a textului cum ar fi editare de text, căutare a erorilor de limbă, căutare și înlocuire de text, schimbare a *font*-ului, copiere și lipire de text, vizualizare pentru tipărire, inserare de hiperlegături, formatare a fundalului, formatare de tabele sau deschidere și editare de pagini *Web* în format *HTML*.

Microsoft Excel 2000

Operațiile *Excel* efectuate constau din închiderea și deschiderea de foi de calcul, pagini *HTML* sau fișiere text, căutarea erorilor de limbă, editare, calcul cu formule, desenarea de grafice sau histograme, formatare de grafice sau celule, atribuirea unor denumiri pentru anumite celule și inserarea de hiperlegături.

Microsoft PowerPoint 2000

Operațiile efectuate de *script*-ul pentru *PowerPoint* includ deschiderea și închiderea de *slide*-uri *PowerPoint* și pagini *HTML*, căutarea erorilor de limbă, editare, formatare și mutare de imagini, aplicarea de șabloane, formatarea tabelelor din *slide*-uri, inserarea de hiperlegături, inserarea de antete și subsoluri pentru pagini, formatarea și rotirea graficelor, aplicarea de efecte grafice și sonore și adăugarea fișierelor video.

Dragon NaturallySpeaking Preferred 4.0

Aceasta este o aplicație de recunoaștere vocală folosită pentru a converti vocea umană în text. *Script*-ul încarcă un fișier `.WAV` folosind utilitarul *Dragon's PlayWave* și apoi cere convertirea fișierului în fișier text.

Netscape Communicator 4.61

- Se deschide un *site Web* care are ca subiect piesele lui *Shakespeare*, iar apoi se selectează și se încarcă textul de câteva ori.
- Se încarcă o pagină care conține multe imagini și tabele și se vizualizează imaginile din această pagină.
- Se încarcă un număr mare de pagini care conțin imagini, tabele și texte și sunt efectuate mai multe căutări de text în interiorul acestor pagini.

Adobe Premierre 5.1

- Se preiau câteva imagini, *clip*-uri video și *clip*-uri audio și, cu ajutorul lor, se creează un film.
- Se creează o animație folosind 16 fișiere `.BMP` și câteva *clip*-uri `.AVI`.

- Se adăugă două *track*-uri audio și se aplică anumite filtre de sunet.
- Filmul obținut în final este salvat în format `.AVI`.

Adobe Photoshop 5.5

Operațiile efectuate de *script*-ul corespunzător acestei aplicații sunt încărcarea și redimensionarea unor imagini, aplicarea diferitelor filtre, modificarea setărilor de culoare, modificarea luminozității sau a contrastului și salvarea de imagini în format `.JPG` optimizat pentru publicarea pe *Web*.

Avid Elastic Reality 3.1

Aceasta este o aplicație de procesare a imaginilor folosită pentru a realiza metamorfoza unei imagini într-o alta. Sunt încărcate două imagini cu dimensiunea de 750 KB, este definit modul în care va fi realizată metamorfoza, iar apoi este redată transformarea primei imagini în a doua.

Metacreation's Bryce 4

Această aplicație este folosită pentru a crea scene fixe sau animate care conțin obiecte reale sau imaginare. Aceste imagini pot fi redate folosind efecte realiste de lumini și umbre. De asemenea, în scenele animate, interacțiunea dintre obiecte (formate din diverse materiale) este redată foarte realistic. *Script*-ul preia o astfel de scenă și redă imaginea finală, după care deschide o nouă imagine și o salvează în formatul video *Metastream* care poate fi folosit pentru redarea de secvențe video pe *Internet*. În final, *script*-ul deschide o nouă imagine și o salvează în format *HTML*.

Microsoft Windows Media Encoder 4.0

Acest produs este folosit pentru a codifica secvențe video și audio în format *ASF* (*Advanced Streaming Format*). Rezultatul operațiilor poate fi ascultat sau vizualizat folosind *Microsoft Windows Media Player*. *Script*-ul preia un *clip* `.AVI` și îl transformă în format `.MPEG-4`.

Rezultatele testului

În cele ce urmează vom prezenta rezultatele *SYSmark 2000* obținute de majoritatea procesoarelor disponibile pe piață. Pentru a măsura corect performanțele, sistemul testat trebuie să păstreze, pe cât posibil, caracteristicile sistemului etalon.

Datorită faptului că dorim să măsurăm performanțele procesoarelor vom modifica, în majoritatea cazurilor, doar placa de bază, tipul și frecvența procesorului.

Pentru a prezenta și rentabilitatea procesoarelor vom include și prețurile la care acestea puteau fi achiziționate în Statele Unite, la sfârșitul lunii februarie.

Intel Pentium III

Vom începe cu această clasă de procesoare, deoarece singura modificare față de sistemul etalon constă în placa de bază (care are magistrala de 133 MHz) și în frecvența procesorului.

Rezultatele obținute sunt următoarele:

Pentium III 533	*126 puncte*	*100$*
Pentium III 600	*137 puncte*	*104$*
Pentium III 667	*148 puncte*	*113$*
Pentium III 733	*157 puncte*	*145$*
Pentium III 800	*168 puncte*	*170$*
Pentium III 866	*177 puncte*	*179$*
Pentium III 933	*186 puncte*	*219$*
Pentium III 1000	*195 puncte*	*254$*

Vom analiza acum scalabilitatea acestui tip de procesoare. În principiu, un procesor este scalabil dacă o creștere a frecvenței cu un anumit procent duce la o creștere a performanțelor cu un procent apropiat.

Pentium III 533	*126*
Pentium III 600 (+12.50%)	*137 (+8.73%)*
Pentium III 667 (+25.50%)	*148 (+17.46%)*
Pentium III 733 (+37.50%)	*157 (+24.60%)*
Pentium III 800 (+50.00%)	*168 (+33.33%)*
Pentium III 866 (+62.50%)	*177 (+40.48%)*
Pentium III 933 (+75.00%)	*186 (+47.62%)*
Pentium III 1000 (+87.50%)	*195 (+54.76%)*

Pentru a completa caracterizarea familiei de procesoare *Pentium III* vom prezenta acum și raportul preț/performanță, adică vom vedea cât ar trebui plătit pentru un punct *SYSmark 2000*. De asemeneam, vom arăta și cât se plătește pentru fiecare megahertz.

Pentium III 533	*0.793$/punct*	*0.188$/MHz*
Pentium III 600	*0.759$/punct*	*0.173$/MHz*
Pentium III 667	*0.764$/punct*	*0.169$/MHz*
Pentium III 733	*0.924$/punct*	*0.198$/MHz*
Pentium III 800	*1.012$/punct*	*0.213$/MHz*
Pentium III 866	*1.011$/punct*	*0.207$/MHz*
Pentium III 933	*1.177$/punct*	*0.235$/MHz*
Pentium III 1000	*1.303$/punct*	*0.254$/MHz*

Cel mai bun raport apare în dreptul procesorului Intel *Pentium III 600*. Totuși, această frecvență este relativ mică, motiv pentru care la 600 MHz se recomandă un procesor *Duron* sau *Celeron*. Pentru familia de procesoare *Pentium III* cea mai bună alegere în acest moment pare a fi frecvența de 866 MHz.

Intel Pentium 4

Vom trece acum la familia de vârf a procesoarelor *Intel*. Pentru a putea testa aceste procesoare va trebui să modificăm atât placa de bază cât și memoria din sistemul etalon. Rezultatele sunt:

Pentium 4 1300	*188 puncte*	*342$*
Pentium 4 1400	*199 puncte*	*350$*
Pentium 4 1500	*205 puncte*	*569$*

Se observă faptul că performanțele procesoarelor *Pentium 4* cu frecvența de 1,3 GHz sunt inferioare performanțelor procesoarelor *Pentium III* cu frecvența de 1 GHz.

Vom studia acum scalabilitatea acestui tip de procesoare:

Pentium 4 1300	*188*
Pentium 4 1400 (+7.69%)	*199 (+5.85%)*
Pentium 4 1500 (+15.38%)	*205 (+9.04%)*

Vom prezenta acum și raportul preț/performanță pentru această familie de procesoare.

Pentium 4 1300	*1.819$/punct*	*0.263$/MHz*
Pentium 4 1400	*1.759$/punct*	*0.250$/MHz*
Pentium 4 1500	*2.776$/punct*	*0.380$/MHz*

Este evident faptul că, în acest moment, cea mai bună alegere o constituie procesorul *Pentium 4* cu frecvența de 1,4 GHz. Totuși, performanțele acestor procesoare nu justifică prețurile foarte mari practicate de *Intel*.

Intel Celeron

Procesoarele *Celeron* au performanțe reduse față de cele din familia *Pentium III* datorită faptului că magistrala lucrează la o frecvență mai redusă și cantitatea de memorie *cache* este mai mică. Iată cum s-au comportat aceste procesoare în testul *SYSmark 2000*:

Celeron 533	*92 puncte*	*57$*
Celeron 566	*96 puncte*	*54$*
Celeron 600	*100 puncte*	*61$*
Celeron 633	*106 puncte*	*59$*
Celeron 667	*109 puncte*	*69$*
Celeron 700	*111 puncte*	*75$*
Celeron 733	*114 puncte*	*83$*
Celeron 766	*115 puncte*	*110$*
Celeron 800	*126 puncte*	*138$*

Modificarea magistralei de la 66 la 100 MHz pentru procesorul *Celeron 800* a dus la o creștere semnificativă a performanțelor acestuia. Față de precursorul său (*Celeron 766*) acest procesor are o performanță cu 9,57% mai mare, deși frecvența crește cu numai 4,35%.

Chiar și cu magistrala la 100 MHz, un *Celeron* nu se poate compara cu un *Pentium III*, *Celeron 800* nereușind decât să egaleze *Pentium III 533*.

În continuare vom studia scalabilitatea procesoarelor din familia *Celeron*:

Celeron 533	*92*
Celeron 566 (+6.25%)	*96 (+4.35%)*
Celeron 600 (+12.50%)	*100 (+8.70%)*
Celeron 633 (+18.75%)	*106 (+15.22%)*
Celeron 667 (+25.00%)	*109 (+18.48%)*
Celeron 700 (+31.25%)	*111 (+20.65%)*
Celeron 733 (+37.50%)	*114 (+23.91%)*
Celeron 766 (+43.75%)	*115 (+25.00%)*
Celeron 800 (+50.00%)	*126 (+36.96%)*

Scalabilitatea este comparabilă cu cea de la procesoarele *Pentium III*. Mai jos puteți vedea raportul preț/performanță pentru procesoarele *Celeron*.

Celeron 533	*0.620$/punct*	*0.107$/MHz*
Celeron 566	*0.563$/punct*	*0.095$/MHz*
Celeron 600	*0.610$/punct*	*0.102$/MHz*
Celeron 633	*0.556$/punct*	*0.093$/MHz*
Celeron 667	*0.633$/punct*	*0.103$/MHz*
Celeron 700	*0.676$/punct*	*0.107$/MHz*
Celeron 733	*0.728$/punct*	*0.113$/MHz*
Celeron 766	*0.957$/punct*	*0.144$/MHz*
Celeron 800	*1.095$/punct*	*0.173$/MHz*

Se observă că, în general, rapoartele sunt mai mici în cazul procesoarelor *Celeron* decât în cazul procesoarelor *Pentium III*. Pentru familia *Celeron* cea mai bună alegere pare a fi frecvența de 633 MHz.

AMD Duron

Vom începe prezentarea procesoarelor produse de *AMD* cu cele din familia *Duron*. Acestea concurează direct cu procesoarele *Celeron* pentru a domina piața calculatoarelor ieftine. Testul *SYSmark 2000* duce la obținerea următoarelor punctaje:

Duron 600	*131 puncte*	*45$*
Duron 650	*137 puncte*	*46$*
Duron 700	*143 puncte*	*45$*
Duron 750	*148 puncte*	*56$*
Duron 800	*153 puncte*	*67$*
Duron 850	*159 puncte*	*87$*

Este evidentă superioritatea procesoarelor *Duron* față de procesoarele *Celeron*. În principal, acest lucru se datorează frecvenței de 200 MHz a magistralei, față de 66 sau 100 MHz pentru procesoarele *Celeron*.

Iată și o prezentare a scalabilității procesoarelor *Duron*:

Duron 600	*131*
Duron 650 (+8.33%)	*137 (+4.58%)*
Duron 700 (+16.66%)	*143 (+9.16%)*
Duron 750 (+25.00%)	*148 (+12.98%)*
Duron 800 (+33.33%)	*153 (+16.79%)*
Duron 850 (+41.66%)	*159 (+21.37%)*

Scalabilitatea este destul de redusă, fiind mult inferioară celei de la procesoarele *Intel*. În ciuda acestui fapt, aceste procesoare sunt cele mai performante de pe piață, ele având cel mai bun raport preț/performanță în acest moment.

Duron 600	*0.344$/punct*	*0.075$/MHz*
Duron 650	*0.336$/punct*	*0.071$/MHz*
Duron 700	*0.315$/punct*	*0.064$/MHz*
Duron 750	*0.378$/punct*	*0.075$/MHz*
Duron 800	*0.438$/punct*	*0.084$/MHz*
Duron 850	*0.547$/punct*	*0.102$/MHz*

Rapoartele sunt mai mici decât în cazul oricăror procesoare de la *Intel*. Cea mai bună alegere pare a fi, în acest moment, procesorul *Duron 700*.

AMD Athlon/200

Vom prezenta acum caracteristicile procesoarelor *Athlon Thunderbird* cu magistrala de 200 MHz. Rezultatele obținute în testul *SYSmark 2000* sunt următoarele:

Athlon 700	*154 puncte*	*77$*
Athlon 750	*161 puncte*	*72$*
Athlon 800	*168 puncte*	*99$*
Athlon 850	*174 puncte*	*101$*
Athlon 900	*182 puncte*	*122$*
Athlon 950	*187 puncte*	*147$*
Athlon 1000	*193 puncte*	*173$*
Athlon 1100	*205 puncte*	*199$*
Athlon 1200	*216 puncte*	*246$*

Procesoarele *Athlon* au performanțe aproximativ egale cu cele ale procesoarelor *Pentium III*, dar la frecvențe de peste 1 GHz depășesc procesoarele *Pentium 4*.

Vom prezenta acum și scalabilitatea acestor procesoare:

Athlon 700	*154*
Athlon 750 (+7.14%)	*161 (+4.55%)*
Athlon 800 (+14.28%)	*168 (+9.09%)*
Athlon 850 (+21.43%)	*174 (+12.99%)*
Athlon 900 (+28.57%)	*182 (+18.18%)*
Athlon 950 (+35.71%)	*187 (+21.43%)*
Athlon 1000 (+42.86%)	*193 (+25.32%)*
Athlon 1100 (+57.14%)	*205 (+33.12%)*
Athlon 1200 (+71.43%)	*216 (+40.26%)*

Scalabilitatea este superioară celei de la procesoarele *Athlon*, dar mai scăzută decât la procesoarele de la *Intel*.

Pentru a putea concura cu *Intel*, *AMD* a trebuit să mențină un raport preț/performanță atrăgător și pentru procesoarele *Athlon*.

Athlon 700	*0.500$/punct*	*0.110$/MHz*
Athlon 750	*0.447$/punct*	*0.096$/MHz*
Athlon 800	*0.589$/punct*	*0.124$/MHz*
Athlon 850	*0.580$/punct*	*0.119$/MHz*
Athlon 900	*0.670$/punct*	*0.136$/MHz*
Athlon 950	*0.786$/punct*	*0.155$/MHz*
Athlon 1000	*0.896$/punct*	*0.173$/MHz*
Athlon 1100	*0.971$/punct*	*0.181$/MHz*
Athlon 1200	*1.139$/punct*	*0.205$/MHz*

Cel mai bun raport apare în dreptul procesorului *Athlon 750*. Totuși, această frecvență este relativ mică și un procesor *Duron* cu o frecvență apropiată este o alegere mai bună. Datorită prețurilor mici ale procesoarelor *Athlon*, merită ahiziționarea unui procesor care atinge bariera de 1 GHz.

AMD Athlon/266

În cursul lunii februarie au apărut pe piață și primele procesoare *AMD* cu magistrala la 266 MHz. Acestea pot fi folosite împreună cu un nou tip de memorie care duce la creșterea semnificativă a performanțelor după cum se poate vedea în continuare:

Athlon 1000	*221 puncte*	*210$*
Athlon 1133	*230 puncte*	*244$*
Athlon 1200	*239 puncte*	*249$*

Este evidentă superioritatea față de procesoarele cu frecvențe echivalente, dar cu magistrala la 200 MHz. Superioritatea față de procesoarele *Pentium 4* este foarte clară, chiar și un *Athlon* de acest tip la 1 GHz fiind mai performant decât un *Pentium 4* la 1,5 GHz.

Scalabilitatea este asemănătoare cu cea de la celelalte procesoare de la *AMD*.

Athlon 1000	*221*
Athlon 1133 (+11.33%)	*230 (+4.07%)*
Athlon 1200 (+20.00%)	*239 (+8.14%)*

Datorită faptului că aceste procesoare sunt foarte noi, raportul preț/performanță nu este la fel de atrăgător ca și la celelalte procesoare produse de *AMD*.

Athlon 1000	*0.950$/punct*	*0.210$/MHz*
Athlon 1133	*1.061$/punct*	*0.215$/MHz*
Athlon 1200	*1.042$/punct*	*0.208$/MHz*

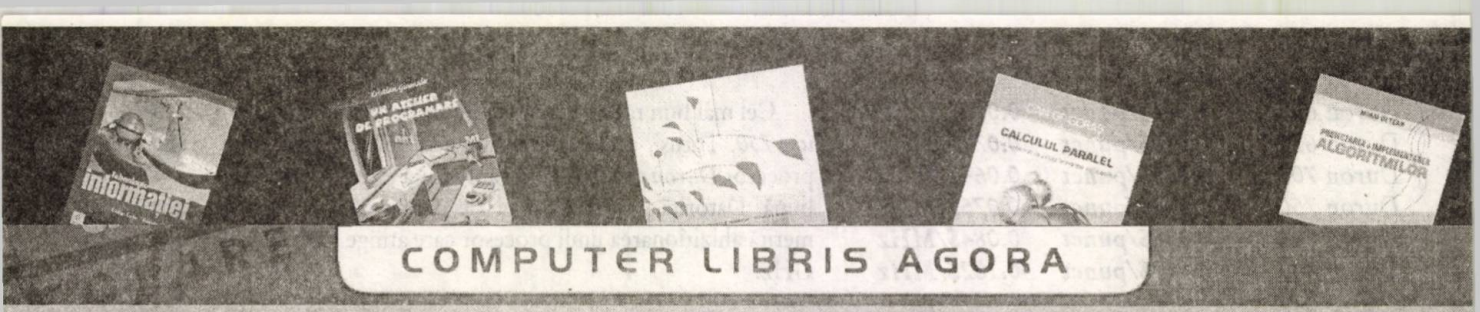
Pentru această familie de procesoare, alegerea cea mai bună pare a fi frecvența de 1 GHz.

În curând

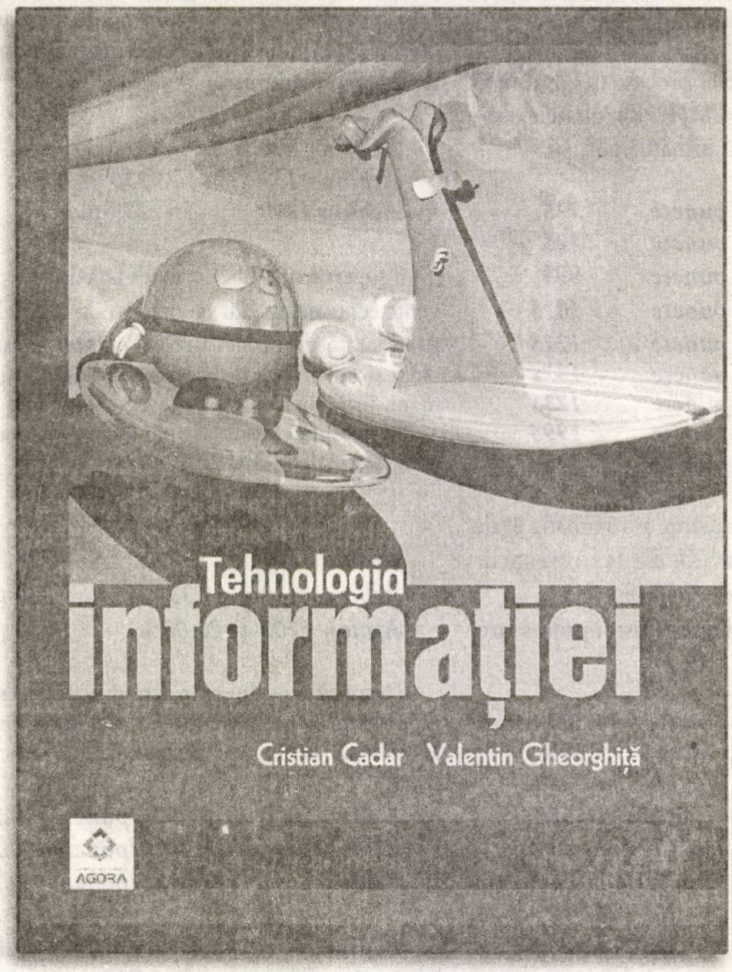
Cel mai așteptat eveniment din perioada următoare îl reprezintă reducerile de prețuri care vor avea loc la începutul lunii martie atât pentru procesoarele *Intel*, cât și pentru cele de la *AMD*. *Intel* a anunțat că următoarea reducere a prețurilor va avea loc pe data de 4 martie, iar *AMD* a anunțat data de 5 martie pentru următoarea reducere. Aceste date pot fi modificate, dar noile reduceri sunt iminente, chiar dacă vor mai întârzia puțin.

De asemenea, este așteptată lansarea procesorului *Celeron 850* și a procesorului *Athlon 1300*.

Mihai Scorțaru este redactor-șef adjunct al GInfo. Poate fi contactat prin e-mail la adresa **skortzy@xnet.ro**.



COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

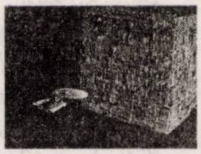
Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Avocații MICROSOFT

Carmen Șușman



Vom prezenta acum un nou dialog imaginar care ar putea avea loc pe puntea navei stelare USS Enterprise în timpul unei întâlniri neplăcute cu borgii. Veți vedea cum poate fi obținută victoria folosind o tehnologie a secolului XX.

Cpt. Jean Luc Picard: *Domnule LaForge, s-a înregistrat vreun succes în încercarea de a găsi o slăbiciune în sistemul borgilor? Domnule Data, s-a reușit accesarea comenzilor computerului de pe nava borgilor?*

Lt. cmdr. Geordi LaForge: *Da, căpitane! De fapt, am găsit răspunsul căutând prin arhivele cu date referitoare la calculatoarele de la sfârșitul secolului XX.*

Geordi apasă o tastă și pe ecran apare o imagine.

Lt. cmdr. William Riker (nedumerit): *Ce înseamnă Microsoft?*

Lt. cmdr. Data: *Permiteți-mi să vă explic! Vom trimite acest program, denumit Windows din motive necunoscute, în sistemul borgilor. După ce acesta va ajunge în unitatea lor de comandă, programul va începe să le consume resursele într-un ritm atât de rapid încât nu va mai putea fi oprit.*

Cpt. Jean Luc Picard: *Dar borgii au capacitatea de a se adapta. Nu își vor putea modifica sistemul astfel încât să mărească spațiul de stocare disponibil?*

Lt. cmdr. Data: *Ba da, dar în momentul în care Windows detectează acest lucru, creează o nouă versiune a sa cunoscută sub numele de upgrade. Consumul de resurse va crește exponențial la fiecare iterație. Borgii nu se vor putea adapta suficient de repede. Până la urmă toată capacitatatea lor de procesare va fi preluată de Windows și nu vor mai fi resurse pentru operațiile obișnuite.*

Cpt. Jean Luc Picard: *Excelent! Aceasta este o idee mult mai bună decât cea cu forma geometrică paradoxală.*

15 minute mai târziu

Lt. cmdr. Data: *Am reușit să instalăm programul Windows în unitatea lor de comandă și, conform așteptărilor, a consumat imediat 85% din resurse. Deocamdată nu avem informații referitoare la upgrade-ul așteptat.*

Lt. cmdr. Geordi LaForge: *Senzorii noștri detectează o creștere a capacității de stocare în sistemul borgilor și a capacității unității de execuție pentru a compensa. Totuși, încă nu avem informații referitoare la upgrade-ul care trebuie să apară pentru a compensa această creștere a performanțelor sistemului borgilor.*

Cpt. Jean Luc Picard: *Data, studiază datele istorice pentru a vedea dacă ne-a scăpat ceva!*

Lt. cmdr. Data: *Domnule, cred că există un motiv pentru care nu a apărut upgrade-ul. Se pare că borgii au reușit să evite apariția lui neînregistrând produsul.*

Lt. cmdr. William Riker: *Căpitane, nu avem de ales! Cer permisiunea de a inițializa secvența urgentă de securitate 3F...*

Lt. cmdr. Geordi LaForge (entuziasmat): *Așteptați! Căpitane, am detectat faptul că resursele de pe nava borg au scăzut brusc la 0%.*

Cpt. Jean Luc Picard: *Data, ce arată senzorii acum?*

Lt. cmdr. Data: *Se pare că borgii au descoperit un modul intern al programului Windows numit Solitaire și acest modul folosește întreaga capacitate de procesare.*

Cpt. Jean Luc Picard: *Vom aștepta să vedem cât timp poate acest Solitaire să reducă funcționalitatea sistemului borgilor.*

După două ore

Lt. cmdr. William Riker: *Geordi, care este starea borgilor?*

Lt. cmdr. Geordi LaForge: *După cum ne-am așteptat, borgii încearcă să compenseze necesitățile mărite ale sistemului dar, de fiecare dată când reușesc să îmbunătățească performanțele acestuia, le trimit noi module Win-*

dows care formează un pachet numit Microsoft funpack.

Cpt. Jean Luc Picard: *Cât timp câștigăm dacă situația rămâne neschimbată?*

Lt. cmdr. Data: *Ritmul actual în care borgii reușesc să soluționeze pronblemele care apar îmi permite să prezic un interval de șase ore până când vor reuși să treacă de toate dificultățile.*

Lt. cmdr. Geordi LaForge: *Căpitane, a apărut o nouă navă în acest sector!*

Cpt. Jean Luc Picard: *Identificați nava!*

Lt. cmdr. Data: *Marcajele sunt foarte asemănătoare cu sigla Microsoft.*

În difuzoare: *Vă vorbește amiralul Bill Gates, reprezentant al monopolului Microsoft. Avem confirmarea faptului că în acest sector sunt folosite produse neînregistrate. Pentru a evita complicațiile ar fi bine să vă predați. Aveți 10 secunde!*

Lt. cmdr. Data: *Nava străină a deschis o trapă din care au ieșit sute de obiecte cu formă umanoidă.*

Cpt. Jean Luc Picard: *Mărește imaginea!*

Lt. cmdr. William Riker: *Dumnezeule! Sunt oameni care plutesc direct spre nava borg și nu au costume speciale de susținere a vieții. Cum pot supraviețui în vid?*

Lt. cmdr. Data: *Nu cred că sunt oameni, domnule! Dacă priviți mai atent, veți observa că au în mâini obiecte numite de oamenii secolului XX genți diplomat și sunt îmbrăcați în costume Armani.*

Lt. cmdr. William Riker și cpt. Jean Luc Picard (îngroziți): *Avocați!*

Lt. cmdr. Geordi LaForge: *Nu se poate! Toți avocații au fost distruși în anul 2017, în timpul Marii Deșteptări.*

Lt. cmdr. Data: *Este adevărat, dar se pare că unii au supraviețuit!*

Lt. cmdr. William Riker: *Au înconjurat nava borgilor și o acoperă cu tot felul de hârtii! Îi fac bucățele pe borgi!*

Cpt. Jean Luc Picard: *Stingeți ecranul! Nu pot privi așa ceva. Nici chiar borgii nu merită o astfel de moarte!*

Carmen Șușman este redactor al GInfo. Poate fi contactată prin e-mail la adresa **schummy@email.ro**.

Replicile unui programator klingonian

- Specificațiile sunt pentru cei slabi și timizi!
- Calculatorul acesta nu e bun de nimic! Am nevoie de unul cu două procesoare Pentium III dacă vrei să mă lupt cu acest cod!
- Nu poți aprecia cu adevărat arta programării klingoniene decât dacă citești lucrările în original. Pentru aceasta trebuie să înveți limba klingoniană!
- Indentare?!... Îți voi arăta cum se face o indentare atunci când îți voi indenta craniul!
- Ce-i cu toată vorbăraia despre lansarea pe piață? Klingonienii nu lansează produse pe piață! Software-ul nostru "scapă" liber pe piață și lasă în urmă o dâră sângeroasă de proiectanți și reprezentanți ai companiilor de asigurare!
- Un adevărat războinic klingonian nu își comentează codul!
- Apelurile de funcții realizate de klingonieni nu au parametri. Ele au argumente și aceste argumente sunt întotdeauna cele câștigătoare!
- Depanare?!... Klingonienii nu efectuează depanări ale produselor software! Ele nu sunt destinate celor slabi!
- Am provocat întreaga echipă care se ocupă cu asigurarea calității la un duel de Bat-Leth. Nu ne vor mai deranja niciodată!
- Semnalând această eroare ai atentat la onoarea familiei mele. Pregătește-te să mori!
- Pui la îndoială calitățile codului meu? Ar trebui să te omor chiar aici!
- Utilizatorii vor fi înspăimântați și vor tremura în fața produsului nostru. Lansați-l pe piață imediat! Lansați-l și lăsați-i să fugă ca niște câini ce sunt!

NET REPORT

PC REPORT ÎN MILENIUL III

www.pcreport.ro

Conectat la viitorul digital

COMPUTER PRESS AGORA Str. Tudor Vladimirescu nr. 63/1, Tg. Mures, CP 230/11, 4300

Tel: 065-166516, FAX: 065-166290, e-mail: office@agora.ro

PC MAGAZINE ROMÂNIA

COMPUTER PRESS AGORA

Din acest număr:

- Vedete in 2000
- Internet business
- Premiile de excelență tehnică
- Laborator

www.pcmagazine.ro

Revista utilizatorilor de calculatoare personale

COMPUTER PRESS AGORA

Str. Tudor Vladimirescu nr. 63/1, Tg.Mureș (CP 230/1), 4300
Tel. 065-166516, Fax 065-166290, e-mail: office@agora.ro

Sucursala București: Str. Poterași 35/2 sec.IV (CP 94/49) 75144 București
Tel.: 01-3309282, Fax: 01-3309285

Gazeta de informatică

Revista viitorilor specialiști în tehnologia informației

Aprilie 2001
Vol. 11/4

Lei 15.000

din sumarul numărului:

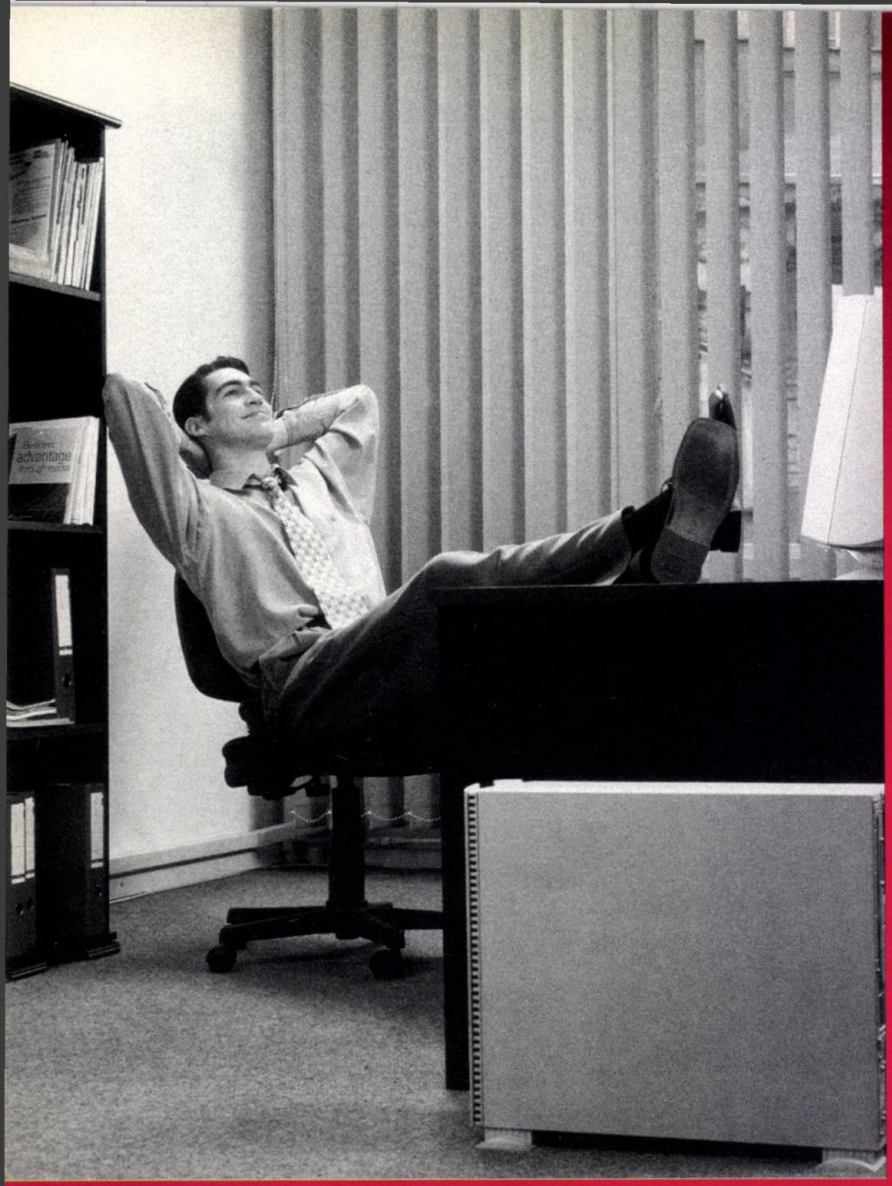
BABEL

INTELIGENTA artificială

TEHNO

AMD revine

SISTEME de fișiere



Erai obișnuit ca lucrurile să stea altfel; de acum însă, ai la dispoziție noul server Compaq ProLiant ML 370 cu procesor Intel® Pentium® III 1.0 GHz. Care își vede singur de treabă și își poartă singur de grijă, astfel ca tu să nu mai duci grija lui. Deoarece Compaq Insight Manager și Integrated Remote Console fac posibilă administrarea serverului de la distanță, în timp ce Compaq Availability Agents monitorizează continuu și repară automat erorile obișnuite care pot apărea. Ceea ce înseamnă mai mult timp pentru tine și pentru ideile tale. Tehnologia care te inspiră. De la Compaq. Bun venit în noua eră IT.

COMPAQ
Inspiration Technology

DE ACUM, AI O POZIȚIE DE INVIDIAT.

COMPAQ PROLIANT ML 370
Procesor Intel® Pentium® III 1.0 GHz • 128 MB RAM (max. 4 GB) • hard disk max. 218 GB hot plug • CD Rom • Fast Ethernet 10/100 WOL • adaptor SCSI Ultra Wide 2

intel inside
pentium® III

TREND
IMPORT - EXPORT
www.trend.ro

Str. Popa Savu nr. 77
Tel: 211 01 90
Fax: 211 01 92
sales@trend.ro

© 2001 Compaq Computer Corporation. COMPAQ and the Compaq logo Registered in U.S. Patent and Trademark Office. ProLiant and Inspiration Technology are trademarks of Compaq Information Technologies Group, LP. in the U.S. and other countries. Intel, the Intel Inside Logo and Pentium are registered trademarks of Intel Corporation.

Vol. 11, Nr. 4 /2001

gazeta de informatică

Sumar

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Redactor:
Carmen Șușman (Alba Iulia)

Au mai colaborat:
Ciprian Ileană (Alba Iulia)
Tiberiu Socaciu (Cluj)
Claudiu Soroiu (Târgu Jiu)

Tehnoredactare:
Mihai Scorțaru

Coperta și grafica:
Andrei Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca

Adresa e-mail: office@ginfo.ro

Tel: 064/192.422
065/166.516
Fax: 065/166.290
http://www.ginfo.ro

ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359

Editor: Computer Press Agora SRL
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihaela Szabo (mszabo@agora.ro)
Ioana Udroiu (iudroiu@agora.ro)

Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

editorial.. PAG. 3

Olimpiada NAȚIONALĂ de INFORMATICĂ

ONI 2001
~ Bacău ~

probleme.. PAG. 5

propuse pentru REZOLVARE

Olimpiada de informatică
~faza județeană~

Arad, Olt, Satu Mare, Suceava, Timiș

concurs.. PAG. 18

Rezultatele rundei a cincea

Clasament

focus.. PAG. 25

O problemă de COMBINATORICĂ

Tiberiu SOCACIU

tehno.. PAG. 29

Sisteme de FIȘIERE

Ciprian ILEANĂ
Claudiu SOROIU

AMD revine

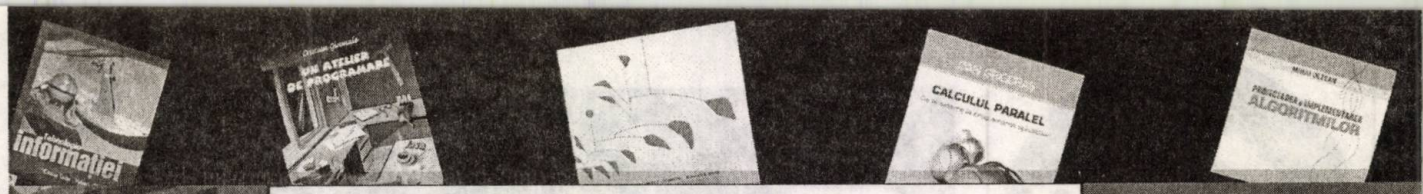
Mihai SCORȚARU

babel.. PAG. 40

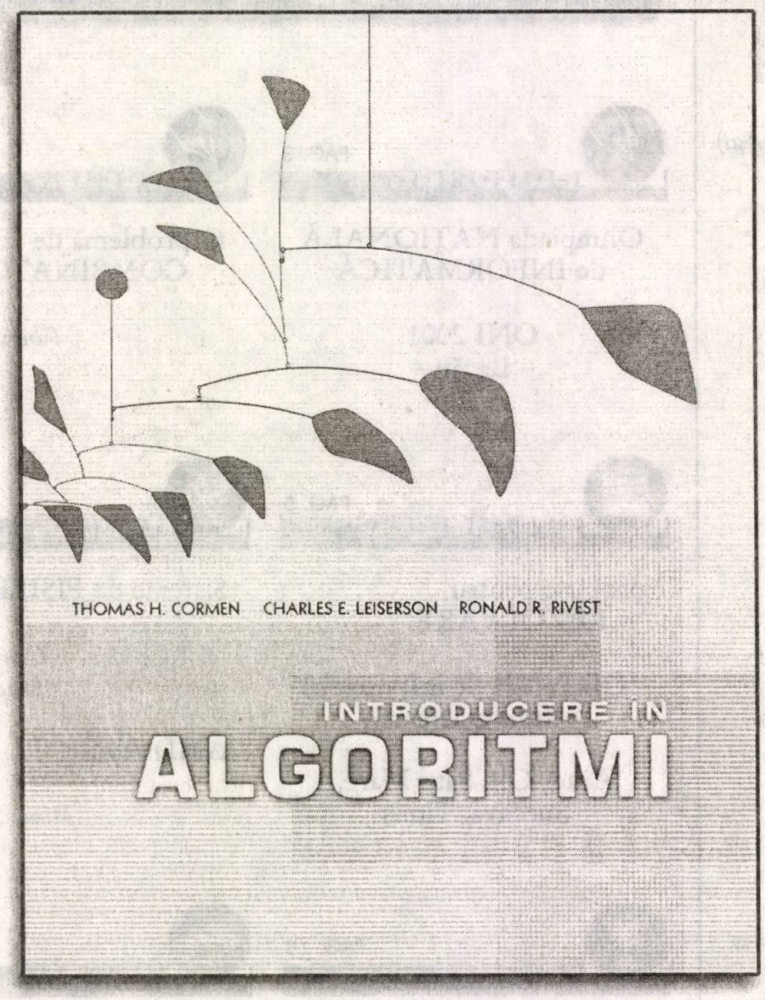
Inteligența ARTIFICIALĂ

Mihai SCORȚARU

Sărbători fericite



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Olimpiada NAȚIONALĂ de INFORMATICĂ

Probabil că în momentul în care citiți aceste rânduri, *Olimpiada Națională de Informatică* este în plină desfășurare sau poate chiar s-a încheiat.

Scriu acest editorial cu câteva zile înaintea plecării spre *Bacău*, orașul în care are loc, anul acesta, întâlnirea celor mai talentați elevi pasionați de programare.

Răsfoiesc numerele vechi ale revistei, apărute în anii trecuți și descopăr dintr-odată că după 1990, majoritatea olimpiadelor naționale au fost coordonate de *colaboratori sau membri ai redacției noastre*.

Anul acesta președintele *Olimpiadei Naționale de Informatică* va fi *dl. prof. dr. Horia Georgescu*, care (nu întâmplător) este și coordonatorul științific al *Gazetei de Informatică*.

Ne face deosebită plăcere să constatăm că mai mult de trei sferturi dintre membrii comisiei de la *ONI 2001* au publicat articole în *GInfo*. Și alt motiv de bucurie: ca de fiecare dată, olimpiada va fi un prilej de întâlnire, de data aceasta nu doar pe paginile revistei, ci personal a celor care au făcut ca această revistă să devină ceea ce este acum.

Nivelul de dificultate al problemelor de olimpiadă a crescut foarte mult în ultimii ani. De exemplu, o problemă propusă spre rezolvare la o olimpiadă internațională de la începutul anilor '90 a devenit acum un banal subiect de teză.

Mai mulți medaliați ai primelor ediții ale olimpiadelor internaționale au recunoscut că, dacă nivelul propriilor cunoștințe ar fi rămas același ca în momentul în care obțineau acele medalii, acum nu ar mai fi siguri că ar reuși să se califice la faza națională a olimpiadei de informatică.

Îmi aduc aminte că în 1996, la olimpiada de la Suceava, metoda programării dinamice era cunoscută doar de câțiva participanți. Acum, chiar și elevii de clasa a IX-a folosesc această metodă, iar un elev de clasa a X-a are puține șanse să obțină un premiu dacă nu o știe aplica la rezolvarea problemelor.

Este normal să ne întrebăm care este explicația acestei evoluții surprinzătoare. Cu siguranță, *GInfo* a avut contribuția sa. Până acum câțiva ani reprezenta singura modalitate organizată de a afla enunțurile și rezolvările problemelor de la diferitele concursuri naționale și internaționale. De asemenea, articolele din revistă prezentau diverse metode și tehnici de rezolvare a problemelor.

Și acum se apropie din nou olimpiada. Evident, în numărul următor veți găsi enunțurile tuturor problemelor de la *ONI 2001*. Probabil bănuiți că în spatele acestor enunțuri se află foarte multă muncă. Mai întâi, autorii lor le-au gândit, le-au formulat și le-au rezolvat. Apoi, *dl. prof. dr. Horia Georgescu* le-a strâns și le-a studiat cu mult timp înainte de începerea olimpiadei. Timp de câteva săptămâni, el și colaboratorii săi, au dat enunțurilor forma pe care acestea o au în momentul în care le primesc concurenții.

Din păcate, claritatea enunțurilor de la olimpiadele județene nu se compară cu cea a problemelor de la olimpiada națională. În acest număr veți găsi și câteva dintre enunțurile problemelor de la olimpiadele județene din diferite zone ale țării. Unele, din nefericire, au trebuit modificate pentru a deveni mai clare.

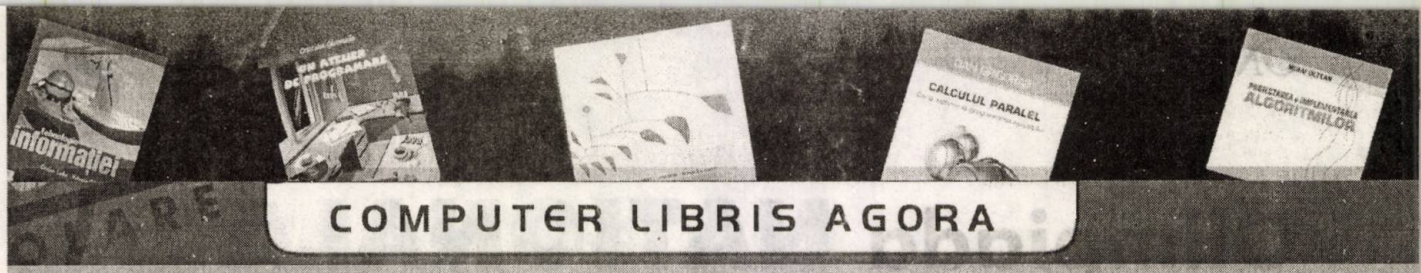
Sperăm că la faza națională enunțurile vor fi lipsite de ambiguități, cerințele clar punctate și limitele datelor de intrare corect specificate. Și acest lucru se datorează, în mare parte, domnului profesor *Georgescu* și echipei sale.

Gazeta de Informatică s-a aflat, încă de la începutul existenței sale, alături de cei care se ocupau de olimpiadele de informatică și, ne place să credem, nu i-a încurcat prea mult. Sperăm că și în următorii ani *GInfo* va avea rolul său în pregătirea concurenților pentru aceste competiții. *Trebuie doar să ni se permită să facem acest lucru.*

Experiența dobândită vorbește de la sine. Cele două ediții ale *Olimpiadei de Informatică a Europei Centrale* desfășurate în România au fost coordonate de membri ai redacției *GInfo*. Și nu am amintit de rolul concursului *CPA*, care permite multora să nu își "iasă din mână".

Nu pot să închei fără să le urez succes tuturor participanților la *Olimpiada Națională de Informatică* de la *Bacău*.

Mihai Scorțaru



Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Încercați-vă puterile!

PROBLEME propuse pentru REZOLVARE

În luna martie a avut loc faza județeană a Olimpiadei de Informatică. În cadrul acestui număr vă prezentăm enunțurile problemelor propuse spre rezolvare în cinci județe ale țării.

P040101: Baza

Arad, clasa a IX-a

Se dau două numere întregi X și Y și se cere să se determine cea mai mică bază pentru X și o bază pentru Y astfel încât numerele X și Y reprezentate în cele două baze să reprezinte aceeași valoare dacă le citim în baza 10. Dacă există mai multe valori ale bazei lui Y care satisfac condiția, se va alege valoarea minimă. Să considerăm numerele `12` și `5`. Evident, considerate în baza `10`, aceste numere nu sunt egale. Dar, dacă `12` este considerat scris în baza `3` iar `5` ca un număr în baza `6`, avem $12_{(3)}=5_{(10)}$, $5_{(6)}=5_{(10)}$; deci cele două numere sunt egale. Cu alte cuvinte `12` și `5` sunt egale dacă selectăm corect bazele în care sunt scrise.

Intrare

Datele de intrare se vor citi din fișierul **`INPUT.IN`**. Pe fiecare linie a fișierului de intrare se află o pereche de numere întregi X, Y separate prin cel puțin un spațiu și care conțin cel mult 80 de cifre. Bazele asociate numerelor X și Y sunt cuprinse între `2` și `36` (inclusiv) și nu sunt neapărat egale. În reprezentarea numerelor, cifrele `0`, `1`, ..., `9` au interpretarea uzuală, iar literele `A`, `B`, ..., `Z` reprezintă cifrele cuprinse între `10` și `35`. Toate datele sunt corecte.

Ieșire

Pentru fiecare pereche de numere întregi de la intrare, se va scrie în fișierul **`OUTPUT.OUT`** o linie care va conține în ordine: `bazax`, `bazay`, separate prin cel puțin un spațiu. Dacă nu există soluție, linia va conține două valori zero (`0 0`).

Timp de execuție: 1 secundă/test

Exemplu

`BAZA.IN`	**`BAZA.OUT`**
`12 5`	`3 6`
`10 A`	`10 11`
`12 34`	`17 5`
`123 456`	`0 0`
`1k 2`	`0 0`
`10 2`	`2 3`

P040102: Numărul care lipsește

Arad, clasa a IX-a

Se dă un număr n și un șir de n - 1 numere naturale distincte cuprinse între `1` la n. Să se afișeze numărul care lipsește.

Intrare

Datele de intrare se citesc din fișierul **`NUM.IN`**. Pe prima linie a acestui fișier se află numărul n ($1 \le n \le 10000000$), iar pe următoarele n - 1 linii se vor afla cele n - 1 numere.

Ieșire

Fișierul de ieșire **`NUM.OUT`** va conține numărul care lipsește.

Timp de execuție: 60 secunde/test

Exemplu

`NUM.IN`	**`NUM.OUT`**
`6`	`4`
`6`	
`1`	
`2`	
`3`	
`5`	

P040103: Numere puternice

Arad, clasa a IX-a

Numerele puternice sunt acele numere naturale care au numărul divizorilor mai mare decât numărul divizorilor oricărui număr natural mai mic decât numărul respectiv. Cel mai mic număr puternic este `1`; urmează `2`, `4`, `6` etc. Scrieți un program care determină al n-lea număr puternic ($n < 70$). Numărul n se citește din fișierul **`PUTERNIC.IN`**, iar ieșirea va apărea în fișierul **`PUTERNIC.OUT`**.

Timp de execuție: 1 secundă/test

Exemplu

`PUTERNIC.IN`	**`PUTERNIC.OUT`**
`3`	`4`

P040104: Raport

Arad, clasa a IX-a

Se dau două numere naturale M și N ($0 < M < N \leq 100000$). Se cere să se tipărească numărul M/N în formă zecimală completă (cu perioadă, dacă este cazul).

Datele de intrare se citesc din fișierul **RAPORT.IN** care conține pe o singură linie numerele M și N, despărțite printr-un spațiu.

Rezultatul se va tipări în fișierul **RAPORT.OUT** pe o singură linie sub forma

`0.parte_neperiodica`

sau

`0.(parte_periodica)`

sau

`0.parte_neperiodica(parte_periodica)`

după cum este cazul.

Timp de execuție: 1 secundă/test

Exemple

RAPORT.IN	RAPORT.OUT
13 25	0.52

RAPORT.IN	RAPORT.OUT
73 505	0.1(4455)

RAPORT.IN	RAPORT.OUT
1 3	0.(3)

P040105: Printre puncte

Olt, clasa a IX-a

Se dă o rețea pătratică de $n \times n$ puncte, astfel încât distanța dintre două puncte vecine pe orizontală sau verticală este identică.

În fiecare punct al rețelei se află un cartonaș pe care este scrisă o literă. Nu se face distincție între literele mari și literele mici.

a) În rețea se formează dreptunghiuri având laturile paralele cu marginile rețelei, dreptunghiuri care au ca vârfuri punctele care conțin același caracter în colțuri. Un punct se consideră că face parte din dreptunghiul format respectând cerințele problemei, dacă se află pe marginile acestuia, în interiorul lui sau în colțuri. Să se afișeze coordonatele colțurilor stânga jos și dreapta sus pentru acele dreptunghiuri care conțin cel mai mare număr de caractere care se repetă (de exemplu, dacă există în dreptunghi patru caractere `'a'`, șapte caractere `'c'` și un caracter `'d'`, atunci numărul caracterelor care se repetă este `4 + 7 = 11` caractere).

b) Unele puncte din rețea nu pot fi incluse în dreptunghiuri, altele fac parte din mai multe dreptunghiuri. Să se precizeze numărul punctelor din rețea care nu fac parte din astfel de dreptunghiuri.

c) Să se determine coordonatele acelor pătrate care se pot forma, nu neapărat cu laturile paralele cu marginile matricei și care nu conțin neapărat același caracter în colțuri, pentru care suma codurilor *ASCII* ale caracterelor din colțuri să fie maximă. Se va afișa suma codurilor maximă pentru pătratele găsite, precum și coordonatele acestora. (Nu contează ordinea de afișare a coordonatelor pătratului.)

Datele de intrare se citesc de la tastatură în ordinea:

- n reprezentând dimensiunea rețelei ($n \leq 10$);
- caracterele rețelei - se va citi de fiecare dată câte o linie întreagă de caractere și între caractere nu există nici un spațiu.

Rezultatele se vor afișa pe ecran.

Timp de execuție: 1 secundă/test

Exemplu

Intrare	*Ieșire*
n=4	a) Nu exista dreptunghiuri.
abcd	b) 16
efgh	c) 438
ijkl	(3,3), (3,4), (4,3), (4,4)
mnop	

P040106: Expresie

Olt, clasa a IX-a

Se dă un șir de caractere, cu maxim 255 de caractere. Să se determine cea mai lungă expresie corectă matematic din punct de vedere sintactic, obținută prin ștergere de caractere din șirul inițial.

Expresia nu conține decât operatorii binari: +, -, *, /. Un operand din expresie este o literă mică sau o literă mare. Expresia nu conține paranteze.

Dacă există mai multe expresii corecte matematic de aceeași lungime maximă, se va afișa numai una singură.

Șirul de caractere se citește de la tastatură, iar expresia corectă matematic va fi afișată pe ecran.

Timp de execuție: 1 secundă/test

Exemplu

Intrare	*Ieșire*
a+b-c*+d	a+b-c*d

P040107: Stație

Satu Mare, clasa a IX-a

Se dă o autostradă circulară cu n stații de benzină, numerotate de la 1 la n. O mașină parcurge tot traseul într-un singur sens, pornind de la o stație dată.

Să se afișeze de câte ori poate să parcurgă mașina traseul dat, dacă la fiecare stație de benzină poate procura combustibil. Dacă traseul dat nu poate fi parcurs nici măcar o dată se va afișa mesajul "*Traseul nu poate fi parcurs*".

Datele de intrare se citesc din fișierul **STATIE.TXT** care are următoarea structură:

- pe prima linie a fișierului se află un număr care reprezintă numărul de stații și unul care indică numărul de kilometri care pot fi parcurși cu un litru de benzină;

- pe linia a doua se află un număr întreg care reprezintă capacitatea tancului de benzină și un număr care reprezintă stația din care se pleacă;
- pe următoarele linii, pentru fiecare stație de benzină, sunt indicați numărul de litri de benzină care sunt depozitați în stația respectivă și distanța, în kilometri, până la stația următoare.

Pe ecran se va afișa numărul de parcurgeri ale traseului.

Exemplu

STATIE.TXT	*Ieșire (ecran)*
3 10	2
40 2	
10 55	
10 50	
13 45	

P040108: Factorial

Satu Mare, clasa a IX-a

Din fișierul **FACT.IN** se citește un număr natural N ($1000 \le N \le 2000000000$) despre care se știe că reprezintă factorialul unui număr k ($k! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot k$). Scrieți un program care determină și afișează în fișierul **FACT.OUT** numărul k.

Exemple

FACT.IN	FACT.OUT
40320	8

FACT.IN	FACT.OUT
479001600	12

P040109: Numere subprime

Suceava, clasa a IX-a

Se dă un număr natural n nenul format din cel mult nouă cifre. Să se determine toate numerele prime distincte care se pot forma cu cifre aflate pe poziții consecutive ale numărului dat.

Fișierul de intrare **NUMAR.IN** conține numărul dat n, iar fișierul de ieșire **NUMAR.OUT** va conține toate numerele prime distincte care pot fi formate folosind cifre ale numărului n aflate pe poziții consecutive.

Timp de execuție: 2 secunde/test

Exemplu

NUMAR.IN	NUMAR.OUT
131	3 13 31 131

P040110: Cazemate

Suceava, clasa a IX-a

Un dispozitiv de apărare este format din n ($1 \le n \le 10000$) cazemate așezate în linie și numerotate de la stânga la dreapta cu $1, 2, 3, \ldots, n$. Pe un aliniament paralel cu acesta se deplasează două tancuri care încearcă să distrugă cazematele.

Primul tanc pleacă din dreptul cazematei 1 și se deplasează până în dreptul cazematei n, apoi se întoarce și așa mai departe.

Al doilea tanc pleacă din dreptul cazematei n și se deplasează până la cazemata 1, apoi se întoarce și așa mai departe.

Fiecare tanc parcurge spațiul dintre două cazemate consecutive într-o secundă.

Primul tanc trage o dată la p ($1 \le p \le 500$) secunde, iar al doilea tanc trage o dată la q ($1 \le q \le 500$) secunde, fiecare în cazemata în dreptul căreia se află.

Tancurile pot trage simultan în aceeași cazemată sau într-o cazemată deja distrusă.

Se cere să se determine timpul în care cele două tancuri distrug toate cazematele. Dacă tancurile nu pot distruge cazematele, în fișierul de ieșire se va scrie 0.

Prima linie a fișierului de intrare **CAZEMATE.IN** conține numărul n al cazematelor, iar a doua linie conține numerele p și q care reprezintă intervalul de timp care trece între două tiruri ale celor două tancuri.

Fișierul de ieșire **CAZEMATE.OUT** va conține intervalul de timp care va trece până în momentul în care toate cazematele vor fi distruse sau valoarea 0 dacă nu este posibilă distrugerea lor.

Timp de execuție: 1 secundă/test

Exemplu

CAZEMATE.IN	CAZEMATE.OUT
5	9
2 3	

P040111: Iceberg

Timiș, clasa a IX-a

Descriem un iceberg cu ajutorul unei matrice. Punctele marcate cu gri reprezintă pozițiile aparținând iceberg-ului. Dacă asupra lui suflă un vânt cald, el începe să se topească de pe margini spre interior.

Regula topirii este următoarea: într-un interval de timp se topește acea porțiune de gheață care are cel puțin în două vecinătăți aer (notate cu 1). Astfel se produc alte astfel de câmpuri (notate cu 2), care se vor topi în al doilea interval de timp ș.a.m.d.

Să generalizăm această problemă pentru trei dimensiuni, considerând iceberg-ul format prin suprapunerea a K astfel de matrice (în loc de pătrate lucrăm cu cuburi). În acest caz se va topi acea bucată de ghiață, care din cele șase vecinătăți are în cel puțin trei aer.

Scrieți un program care pentru un iceberg dat, returnează în câte intervale de timp se topește întreg iceberg-ul, respectiv pentru fiecare interval câte câmpuri de ghiață mai are iceberg-ul.

Intrare

Numele fișierului de intrare se citește de la tastatură. Pe prima linie apar trei numere întregi, separate printr-un spa-

țiu, care reprezintă numărul de linii ($1 \le N \le 40$), numărul de coloane ($1 \le M \le 40$) și numărul de plane ($1 \le K \le 40$) ale iceberg-ului. Urmează descrierea celor K plane. Fiecărui plan îi corespund N linii și pe fiecare linie sunt M cifre egale cu valoarea 0 dacă este aer sau valoarea 1 dacă este gheață pe acea poziție. Pe margini este sigur aer.

Ieșire

Numele fișierului de ieșire se citește de la tastatură. Acesta va conține pe prima linie numărul de unități de timp în care se va topi toată gheața (T). Următoarele T linii vor conține pe fiecare linie i numărul de unități de gheață existente la începutul intervalului de timp i. (La ultimul interval de timp vor exista 0 bucăți de gheață; această informație nu va mai fi scrisă în fișier.)

Exemplu

ICEBERG.IN	ICEBERG.OUT
4 5 4	2
00000	12
00000	4
00000	
00000	
00000	
01110	
01110	
00000	
00000	
01110	
01110	
00000	
00000	
00000	
00000	
00000	

P040112: Concurs de dans

Timiș, clasa a IX-a

Un ansamblu are n fete și m băieți ($1 \le n \le m \le 25$), unii profilați pe dans modern, alții pe dans clasic.

În cadrul unui concurs, fiecărui component al ansamblului, în funcție de performanțele acestuia, i se atribuie un punctaj (diferit de al celorlalți) din intervalul (0, 25].

Determinați n perechi (fată, băiat), astfel încât echipa să obțină punctaj maxim știind că pentru o pereche la care ambii concurenți sunt profilați pe același tip de dans, punctajul echipei va crește cu produsul punctajelor celor doi concurenți, iar dacă perechea este formată din concurenți profilați pe dansuri diferite, punctajul echipei va scădea cu produsul punctajelor celor doi concurenți. Să se afișeze punctajul maxim al echipei și perechile care se prezintă în concurs.

Intrare

Numele fișierului de intrare se citește de la tastatură. Pe prima linie se află numărul n al fetelor, iar pe următoarele n linii se află informații despre punctaj, numărul de concurs și tipul de dans practicat de fiecare fată.

Pe următoarea linie se află numărul m al băieților, iar pe următoarele m linii se află informații despre punctaj, numărul de concurs și tipul de dans practicat de fiecare băiat.

Numărul de concurs este un număr natural format din maxim două cifre, iar tipul de dans este codificat cu un caracter: "m" pentru dans modern și "c" pentru dans clasic.

Ieșire

Datele de ieșire se vor scrie într-un fișier text al cărui nume se va citi de la tastatură. Fișierul va conține, pe câte o linie, perechi de forma (*nr_concurs_fată*, *nr_concurs_băiat*). Pe ultima linie se va scrie punctajul echipei.

Timp de execuție: 40 secunde/test

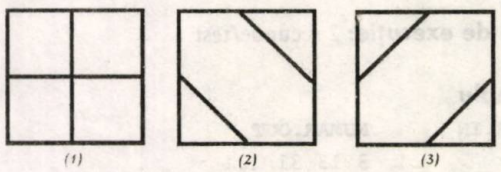
Exemplu

DANS.IN	DANS.OUT
6	(16,16)
2 21 c	(21,29)
6 31 c	(31,25)
10 64 c	(15,33)
8 62 c	(62,21)
7 15 c	(64,18)
1 16 c	287
9	
3 17 c	
9 21 c	
8 33 c	
7 25 c	
4 24 c	
1 62 c	
5 16 c	
10 18 c	
6 29 c	

P040113: Black

Arad, clasa a X-a

Se consideră o tablă de 5×5 pătrățele și 25 de piese pătrate, fiecare piesă putând avea una din formele:



Se observă că piesa *(1)* are conectate marginile N-S și E-V, piesa *(2)* are conectate marginile N-E și S-V, iar piesa (3) are conectate marginile N-V și S-E. Cele două linii din piesa *(1)* nu se intersectează, ci trec "*una pe sub cealaltă*".

Se cere să se așeze cele 25 de piese pe tablă în așa fel încât să se obțină un drum care:

- să treacă prin fiecare pătrat exact o dată;
- să nu se autointersecteze;

- să pornească din colțul de NV al tablei (linia 1, coloana 1), începând de la exteriorul tablei (fie dinspre nord, fie dinspre vest);
- să se termine în colțul de SE al tablei (linia 5, coloana 5) și să părăsească tabla.

Fișierul de intrare **BLACK.IN** conține, pe o singură linie, trei numere, separate printr-un singur spațiu, reprezentând numărul de piese din tipurile (1), (2) și (3). Suma lor va fi întotdeauna 25.

În fișierul de ieșire **BLACK.OUT** se va tipări o matrice cu 5×5 numere, separate prin spatii, reprezentând tipul piesei plasate în fiecare pătrățel. Dacă există mai multe soluții, se va tipări una la alegere. Dacă nu există nici o soluție, fișierul **BLACK.OUT** va conține numai mesajul `Imposibil`.

Timp de execuție: 1 secundă/test

Exemple

BLACK.IN
```
6 9 10
```

BLACK.OUT
```
1 2 3 1 2
3 3 1 3 3
1 3 3 2 2
1 2 2 3 3
2 1 3 2 2
```

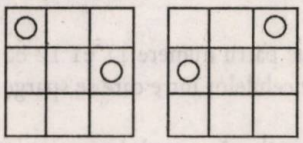
BLACK.IN
```
25 0 0
```

BLACK.OUT
```
Imposibil
```

P040114: Blatistul

Arad, clasa a X-a

În orașul *B.* biletele de autobuz au o grilă de $N \times N$ pătrățele, dintre care aparatele de taxat compostează (găuresc) K pătrățele. Biletul poate fi introdus în aparat numai cu un capăt (celălalt capăt este prins în cotor), dar poate fi introdus fie pe față, fie pe dos. În acest fel, unele din configurațiile posibile de K găuri în grila de $N \times N$ pătrățele sunt oglindiri stânga-dreapta ale altor configurații. Iată un exemplu pentru $N = 3$ și $K = 2$:



(cod: 1123) **(cod: 1321)**

Un blatist veteran face colecție de bilete perforate și dorește să catalogheze toate configurațiile posibile de găuri, ignorând însă oglindirile (deoarece controlorii nu se supără dacă un bilet este taxat pe dos). În acest scop, el codifică fiecare configurație printr-un șir de forma $l_1c_1l_2c_2...l_kc_k$ unde (l_i, c_i) sunt coordonatele găurii i relativ la colțul din stânga-sus al biletului. Găurile sunt enumerate de la stânga la dreapta și de sus în jos. De exemplu, cele două configurații de mai sus sunt codificate prin `"1123"` și `"1321"`. Acum blatistul este capabil să elimine oglindirile după următorul criteriu: dacă o configurație este oglindirea alteia, dintre cele două se va alege numai cea a cărei codificare are o poziție lexicografică mai mică, cealaltă ignorându-se. Dintre cele două configurații de mai sus, `"1123"` are o poziție lexicografică minimă.

Fișierul de intrare **BILET.IN** conține numerele N ($1 \le N \le 8$) și K ($1 \le K \le 3$), separate printr-un singur spațiu.

Fișierul de ieșire **BILET.OUT** va conține, pe fiecare linie, codificarea unei configurații. Configurațiile vor apărea în ordine lexicografică deoarece blatistul trebuie să poată găsi cât mai repede orice bilet.

Timp de execuție: 1 secundă/test

Exemplu

BILET.IN
```
3 2
```

BILET.OUT
```
1112
1113
1121
1122
1123
1131
1132
1133
1221
1222
1231
1232
2122
2123
2131
2132
2133
2231
2232
3132
3133
```

P040115: Fractali

Arad, clasa a X-a

Se consideră pătratul de latură `1` care are colțurile în punctele de coordonate `(0, 0)`, `(1, 0)`, `(1, 1)` și `(0, 1)`. Împărțim acest pătrat în patru părți egale și numerotăm sferturile în ordinea indicată de figura alăturată.

1	2
3	4

Acest procedeu se reia pentru fiecare dintre pătratele obținute; rezultă `16` pătrate, fiecare având atașat câte un cod format din două cifre.

De exemplu, pătratul care are codul `24` are colțurile în punctele de coordonate `(0.75, 0.5)`, `(1, 0.5)`, `(1, 0.75)` și `(0.75, 0.75)`. Configurația obținută este următoarea:

11	12	21	22
13	14	23	24
31	32	41	42
33	34	43	44

Conform aceluiași procedeu fiecare pătrățel se reîmparte, obținând pătrățele care au coduri formate din trei cifre ș.a.m.d. Problema cere ca, indicându-se două pătrate P_1 și P_2 prin codurile lor, să se determine dacă pătratele sunt:

a) identice;
b) P_1 inclus în P_2, fără ca cele două pătrate să aibă un punct comun;
c) P_2 inclus în P_1, fără ca cele două pătrate să aibă un punct comun;
d) P_1 inclus în P_2, astfel încât cele două pătrate să aibă puncte comune;
e) P_2 inclus în P_1, astfel încât cele două pătrate să aibă puncte comune;
f) exterioare unul altuia, dar cu cel puțin două puncte comune;
g) exterioare unul altuia, dar cu un singur punct (vârf) comun;
h) exterioare unul altuia, dar fără să aibă nici un punct comun.

În continuare vă prezentăm câteva exemple pentru fiecare tip de pereche:

- perechi de tipul **a)**: `31` și `31`;
- perechi de tipul **b)**: `141` și `1`;
- perechi de tipul **c)**: `2` și `41`;
- perechi de tipul **d)**: `124` și `1`;
- perechi de tipul **e)**: `4` și `44`;
- perechi de tipul **f)**: `1` și `2`;
- perechi de tipul **g)**: `1` și `4`;
- perechi de tipul **h)**: `11` și `44`.

Fișierul de intrare **`PATRAT.IN`** conține două linii, pe fiecare dintre ele aflându-se codul unui pătrat. Aceste coduri conțin cel mult `10000` de cifre.

Fișierul de ieșire **`PATRAT.OUT`** va conține tipul perechii formate de cele două pătrate.

Tipul se va codifica printr-o literă mare cuprinsă între `A` și `H`.

Timp de execuție: 1 secundă/test

Exemplu

`PATRAT.IN`	`PATRAT.OUT:`
`1444444444`	`F`
`3222222`	

P040116: Numărare de funcții

Arad, clasa a X-a

Să se determine numărul de funcții surjective $f:\{a_1, a_2, ..., a_n\} \to \{-1, 0, 1\}$ cu proprietatea $\sum_{i=1}^{n} |f(a_i)| = s$.

Prima linie a fișierului de intrare **`FUNCTIE.IN`** conține valorile n și s ($1 \le n$, $s \le 50$), separate printr-un singur spațiu.

Fișierul de ieșire **`FUNCTIE.OUT`** va conține o singură linie pe care se va afla numărul funcțiilor care au proprietatea cerută.

Timp de execuție: 1 secundă/test

Exemplu

`FUNCTIE.IN`	`FUNCTIE.OUT:`
`5 3`	`60`

P040117: Speologi

Olt, clasa a X-a

Un grup de speologi a descoperit o peșteră în Munții Apuseni, formată din cel puțin două încăperi care nu comunică între ele. Unul dintre speologi își notează date despre peșteră.

Pentru aceasta, folosește echivalența cu o zonă dreptunghiulară cu m linii și n coloane, cu celule identice și pereți între celule (colțul din stânga sus al zonei are coordonatele `1, 1`).

În fiecare celulă speologul înscrie un număr care se obține astfel: se pornește de la valoarea `0` și se adaugă `1` dacă există perete spre vest, apoi se adaugă `2` dacă există perete spre nord, `4` dacă există perete spre est și `8` dacă există perete spre sud.

La sfârșitul explorării, speologii vor să determine câte încăperi au descoperit, din câte celule este formată cea mai mare încăpere și ce perete trebuie să spargă, astfel încât să se creeze o încăpere de arie maximă, propice vizitatorilor.

Aria unei încăperi este dată de numărul de celule componente.

Datele de intrare se citesc din fișierul **`PESTERA.IN`** astfel:

- prima linie conține valorile m și n ($1 \le m, n \le 10$) reprezentând dimensiunile zonei dreptunghiulare; ele sunt separate printr-un spațiu;
- pe următoarele m linii se află n valori cuprinse între `0` și `15` reprezentând codificările celulelor; ele sunt separate prin spații.

Datele de ieșire se scriu în fișierul **`PESTERA.OUT`** astfel:

- pe prima linie numărul de încăperi;
- pe a doua linie numărul de celule din încăperea cea mai mare;
- pe a treia linie patru numere `l1 c1 l2 c2` reprezentând coordonatele celulelor între care se sparge peretele.

Timp de execuție: 2 secunde/test

Exemplu

`PESTERA.IN`	`PESTERA.OUT`
`3 3`	`9`
`15 15 15`	`1`
`15 15 15`	`1 1 1 2`
`15 15 15`	

P040118: Timbre

Olt, clasa a X-a

Gigel este angajat la Oficiul Poștal din cartier. Zilnic, el trebuie să timbreze plicurile care urmează a fi expediate în străinătate.

În funcție de destinație, pentru fiecare plic în parte, el trebuie să pună timbre de o anumită valoare S ($1 \le S \le 5000$).

Gigel și-a propus să folosească, pentru fiecare plic, cât mai puține timbre. Timbre de aceeași valoare pot fi lipite de mai multe ori pe același plic, respectând condiția ca suma valorilor timbrelor lipite pe plic să fie exact S, iar numărul timbrelor folosite să fie minim.

Dacă nu găsește o combinație de timbre ale căror valori însumate să fie S, plicul respectiv este dat șefului.

La dispoziția lui Gigel există cantități suficiente pentru orice fel de timbru.

Datele de intrare se citesc din fișierul **TIMBRE.IN** astfel:

- prima linie conține numărul n, reprezentând numărul de categorii de timbre disponibile ($n \le 50$); prin categorie se înțelege timbre de aceeași valoare;
- pe următoarea linie se află n valori întregi separate prin spații care reprezentintă valorile timbrelor din fiecare categorie disponibilă. Valoarea maximă a unui timbru este 1000.

Datele de ieșire se vor scrie în fișierul **TIMBRE.OUT** astfel:

- pe prima linie numărul m de timbre lipite de Gigel;
- pe următoarea linie m valori, separate prin spații, reprezentând valorile timbrelor alese.

În cazul în care plicul a fost trimis șefului, fișierul va conține o singură linie, cu valoarea 0.

Timp de execuție: 2 secunde/test

Exemplu

TIMBRE.IN	TIMBRE.OUT
5	4
2 3 5 6 8	2 5 8 8
23	

P040119: Fraze

Satu Mare, clasa a X-a

În fișierul **CUVINTE.IN** se află un număr natural n (pe prima linie), iar pe următoarele n linii se găsesc cuvinte (pe fiecare linie se află un cuvânt despărțit în silabe cu ajutorul caracterului '-').

Cuvintele conțin numai majuscule. Spunem că două cuvinte rimează dacă au cel puțin două litere de la sfârșit identice.

Se cere să se distribuie cuvintele în grupe care rimează și să se afișeze frazele ce conțin număr maxim de silabe și pentru care cuvântul aflat pe poziția i rimeaza cu cel aflat pe poziția $i + 2$, iar cuvintele care rimeaza vor fi ordonate alfabetic. Fiecare frază trebuie să conțină cel puțin trei cuvinte.

Rezultatele vor fi afișate în fișierul **FRAZE.OUT**. Pe fiecare dintre primele linii se vor scrie cuvintele dintr-o grupă separate prin spații (fiecare grupă pe o linie). Urmează o linie vidă și pe următoarele linii se află câte o frază.

Exemplu

CUVINTE.IN	FRAZE.OUT
15	TALISMAN GERMAN UMAN
TA-LIS-MAN	NICI
NICI	GENIUL
GE-NIUL	MACAR DOAR
MA-CAR	LACAT
GER-MAN	UN
LA-CAT	TREI
U-MAN	INFORMATI
DOAR	ALGORITM
UN	CARMEN
TREI	CLARA
IN-FOR-MATI	
AL-GO-RITM	GERMAN DOAR TALISMAN MACAR UMAN
U-NIC	
CAR-MEN	
CLA-RA	

P040120: Teren

Satu Mare, clasa a X-a

Locuitorii unui oraș caută un teren pentru construirea unui aeroport. Ei au la dispoziție harta regiunii.

Această hartă are formă dreptunghiulară și este împărțită în pătrate având lungimea laturii de o unitate.

Fiecare pătrat se identifică printr-o pereche de coordonate (x, y), unde x este coordonata orizontală (vest-est), iar y este coordonata verticală (sud-nord). Pe hartă este indicată altitudinea fiecărui pătrat.

Găsiți o regiune dreptunghiulară de pătrate având aria cea mai mare (adică un teren dreptunghiular format din cel mai mare număr de pătrate) astfel încât:

- cea mai mare diferență dintre pătratul cu cea mai mare altitudine și pătratul cu cea mai mică altitudine de pe teren este mai mic sau egal cu o valoare dată C ($0 \le C \le 10$);
- lățimea (măsurată de-a lungul direcției vest-est) terenului este cel mult 100.

În cazul în care există mai multe terenuri ce satisfac cerințele problemei, se va afișa unul singur.

U și V ($1 \le U, V \le 700$) reprezintă dimensiunile hărții. U indică numărul pătratelor pe direcția vest-est, iar V indică numărul pătratelor pe direcția sud-nord.

Numărul întreg H_{xy} ($-30000 \le H_{xy} \le 30000$) reprezintă altitudinea pătratului având coordonatele (x, y), $1 \le x \le U$, $1 \le y \le V$;

Numele fișierului de intrare este **LAND.INP**. Pe prima linie se află trei numere întregi: U, V și C.

Fiecare dintre următoarele V linii conține câte U numere întregi reprezentând valorile H_{xy} pentru fiecare pătrățel.

Fișierul de ieșire **LAND.OUT** va conține o linie pe care se vor scrie patru numere întregi, localizând terenul determinat: *Xmin*, *Ymin*, *Xmax*, *Ymax*, unde (*Xmin*, *Ymin*) sunt coordonatele pătratului din colțul sud-vest al terenului, iar (*Xmax*, *Ymax*) sunt coordonatele pătratului din colțul nord-est al terenului.

Exemplu

LAND.INP

```
10 15 4
41 40 41 38 39 39 40 42 40 40
39 40 43 40 36 37 35 39 42 42
44 41 39 40 38 40 41 38 35 37
38 38 33 39 36 37 32 36 38 40
39 40 39 39 39 40 40 41 43 41
39 40 41 38 39 38 39 39 39 42
36 39 39 39 39 40 39 41 40 41
31 37 36 41 41 40 39 41 40 40
40 40 40 42 41 40 39 39 39 39
42 40 44 40 38 40 39 39 37 41
41 41 40 39 39 40 41 40 39 40
47 45 49 43 43 41 41 40 39 42
42 41 41 39 40 39 42 40 42 42
41 44 49 43 46 41 42 41 42 42
45 40 42 42 46 42 44 40 42 41
```

LAND.OUT

```
4 5 8 11
```

P040121: Bile

Suceava, clasa a X-a

Într-un sistem ortogonal de coordonate xOy se dau p ($1 \le p \le 1000$) bile prin coordonatele lor. Bilele se deplasează orizontal și vertical. Să se determine poziția k a unei linii orizontale pe care cele p bile să se alinieze una lângă alta, fără goluri între ele, astfel încât deplasarea bilelor până la linia de aliniere să fie minimă. Deplasarea unei bile este dată de numărul de poziții prin care trece aceasta. Două bile nu pot ocupa simultan aceeași poziție după aliniere.

Prima linie a fișierului de intrare **BILE.IN** conține numărul p al bilelor, iar următoarele p linii conțin coordonatele pozițiilor inițiale ale celor p bile. Coordonatele sunt numere întregi cuprinse între 0 și 255.

Fișierul de ieșire **BILE.OUT** va conține pe prima linie poziția k la care se face alinierea și pe a doua linie deplasarea minimă d.

Timp de execuție: 5 secunde/test

Exemplu

BILE.IN	BILE.OUT
11	8
2 3	35
6 6	
6 8	
6 9	
6 10	
6 15	
9 6	
11 9	
13 8	
13 6	
15 9	

P040122: Mărgele

Suceava, clasa a X-a

Se dau n ($1 \le n \le 20$) mărgele de k ($1 \le k \le 9$) culori; m_1 mărgele de culoarea 1, m_2 mărgele de culoarea 2, ..., m_k mărgele de culoarea k.

Să se determine toate modurile posibile de a se așeza cele n mărgele pe un șirag (circular), astfel încât să nu existe două mărgele alăturate de aceeași culoare. O mărgea este considerată prima în șirag, astfel permutările circulare sunt considerate soluții distincte.

De exemplu, pentru trei mărgele de trei culori diferite, 123 și 312 sau 231 sunt considerate soluții distincte și... mai sunt și altele!

Fișierul de intrare **MARGELE.IN** conține pe prima linie numărul n al mărgelelor, iar pe a doua linie valorile m_1, m_2, ..., m_k reprezentând numărul de mărgele din cele k culori.

Fișierul de ieșire **MARGELE.OUT** va conține numărul h al soluțiilor.

Timp de execuție: 10 secunde/test

Exemplu

MARGELE.IN	MARGELE.OUT
4	2
2 2	

Explicație

Cele două soluții din exemplul anterior sunt 1212 și 2121. Acestea nu se vor scrie în fișierul de ieșire.

P040123: Audiență TV

Timiș, clasa a X-a

N ($1 \le n \le 20$) posturi de televiziune fac un sondaj pentru a afla perioadele de maximă audiență pe parcursul unei zile. În acest scop se prelucrează datele culese de la m ($1 \le m \le 1000$) persoane.

Fiecare persoană furnizează numărul de intervale de timp pe care le va specifica și pentru fiecare interval un număr ce indică postul de televiziune și perioada de timp [*ora de început*, *ora de sfârșit*] în care s-a uitat la televizor. O persoană poate indica mai multe astfel de intervale în care a vizionat posturi diferite.

Se cere să se determine pentru fiecare post de televiziune perioadele de audiență maximă precum si postul sau posturile de televiziune cu audiență maximă.

Prima linie a fișierului de intrare **INP.TXT** conține numărul n al posturilor de televiziune și numărul m al telespectatorilor intervievați (cele două numere sunt separate printr-un singur spațiu). Fiecare dintre următoarele m linii conține răspunsurile unui spectator.

Primul număr de pe linie indică numărul p al intervalelor specificate, iar următoarele p grupuri de câte trei numere indică descrierea unui interval. Cele trei numere sunt t, i și s; ele reprezintă postul de televiziune vizionat, ora de începere a vizionării, respectiv ora de terminare a vizionării.

Datele de ieșire se vor scrie în fișierul **OUT.TXT**. Primele *n* linii indică intervalele de audiență maximă pentru fiecare post.

Primul număr identifică postul, al doilea indică numărul telespectatorilor pe care i-a avut atunci când audiența sa a fost maximă și următoarele grupuri de două numere (separate prin virgulă) indică intervalele de timp în care postul a avut audiența maximă.

Ultima linie conține posturile care au avut cea mai mare audiență.

Numerele de pe aceeași linie a fișierului de ieșire sunt separate prin spații.

Posturile sunt numerotate de la 1 la *n*, intervalele de timp sunt specificate folosind numere naturale cuprinse între 0 și 24 și datele de intrare sunt corecte.

Timp de execuție: 1 secundă/test

Exemplu

INP.TXT

```
3 4
3 1 11 15 2 15 20 1 21 24
3 3 10 14 2 15 18 3 20 22
3 1 10 12 2 14 18 1 20 24
2 3 10 15 1 18 20
```

OUT.TXT

```
1 2 11,12 20,20 21,24
2 3 15,18
3 2 10,14
2
```

P040124: Puteri ale lui 2

Timiș, clasa a X-a

Se dă un număr *n* ($1 \le m \le 1000$) și se cere să se scrie acest număr ca sumă (diferență) de oricâte puteri ale lui 2 și ca sumă (diferență) de număr minim de puteri ale lui 2. Expresia va avea forma:

$$n = 2^{k_0} - 2^{k_1} - \ldots - 2^{k_m}$$

Valoarea *n* se va citi de la tastatură. Rezultatele se vor scrie în fișierele **A.TXT** și **B.TXT**.

Prima linie a acestor fișiere va conține numărul de termeni ai expresiei.

Următoarele *m* linii vor descrie câte un termen prin semnul său (+ sau -), urmat de puterea lui 2 corespunzătoare termenului respectiv.

Timp de execuție: 1 secundă/test

Exemplu

Intrare (tastatură)

```
23
```

A.TXT

```
4
+4
+2
+1
+0
```

B.TXT

```
3
+5
-3
-0
```

P040125: Dreptunghi

Arad, clasele a XI-a și a XII-a

Se dă un dreptunghi ale cărui dimensiuni sunt numere naturale cuprinse între 1 și 100.

Scrieți un program care determină o descompunere a acestui dreptunghi în cât mai puține pătrate ale căror laturi să fie paralele cu laturile dreptunghiului. Într-un dreptunghi se poate executa o tăietură completă (de la o margine la alta) paralelă cu o latură. Tăieturile se pot executa și în dreptunghiurile formate, nu numai în dreptunghiul inițial.

Datele de intrare se citesc din fișierul **RECT.IN** care conține, pe o singură linie, dimensiunile *n* și *m* ale dreptunghiului, separate printr-un singur spațiu.

În fișierul de ieșire **RECT.OUT** se va scrie numărul minim de pătrate în care se poate descompune dreptunghiul inițial.

Timp de execuție: 1 secundă/test

Exemple

RECT.IN

```
2 4
```

RECT.OUT

```
2
```

RECT.IN

```
3 2
```

RECT.OUT

```
3
```

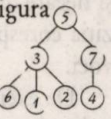
P040126: Parcurgere Euler

Arad, clasele a XI-a și a XII-a

Fie un arbore general cu rădăcina (un nod poate avea oricâți fii). Arborele are *N* noduri numerotate de la 1 la *N*. O parcurgere *Euler* a acestui arbore este realizată astfel:

- se tipărește rădăcina arborelui;
- pentru fiecare dintre fiii rădăcinii se parcurge după aceeași metodă subarborele care are drept rădăcină fiul respectiv și apoi se tipărește rădăcina.

Să considerăm arborele cu șapte noduri din figura alăturată.



Parcurgerea *Euler* a acestui arbore este 5 3 6 3 1 3 2 3 5 7 4 7 5.

Se dă un număr *N* și o succesiune de numere și se cere să se verifice dacă succesiunea reprezintă o parcurgere *Euler* corectă a unui arbore cu *N* noduri și, în caz afirmativ, să se reconstituie arborele.

Fișierul de intrare **EULER.IN** conține pe prima linie valoarea *N* și pe a doua linie o succesiune de numere (cuprinse între 1 și *N*), separate prin spații.

Fișierul de ieșire **EULER.OUT** va conține pe prima linie unul dintre cuvintele DA (dacă succesiunea este o parcurgere *Euler*) și NU (dacă succesiunea nu este o parcurgere *Euler*).

În cazul în care succesiunea este corectă, următoarele *N* - 1 linii vor conține câte două numere, separate prin spații, care vor reprezenta câte o muchie a arborelui. Ordinea de afișare a muchiilor nu contează.

Timp de execuție: 1 secundă/test

Exemple

EULER.IN	EULER.OUT
7	DA
5 3 6 3 1 3 2 3 5 7 4 7 5	5 3
	3 6
	3 1
	3 2
	5 7
	7 4

EULER.IN	EULER.OUT
3	NU
1 2 3 1 2 3 1 2 3 1 2 3	

P040127: Modulo

Arad, clasele a XI-a și a XII-a

Se dau numerele A, B și C ($1 \le A, B, C \le 2000000000$) și se cere calcularea valorii $(A \hat{} B)$ **mod** C (restul împărțirii lui A^B la C).

Cele trei numere se citesc din fișierul **MOD.IN**; A se află pe prima linie, B pe a doua și C pe a treia.

Rezultatul (restul împărțirii lui A^B la C) va fi scris în fișierul **MOD.OUT**.

Timp de execuție: 1 secundă/test

Exemplu

RECT.IN	RECT.OUT
2 4	2

P040128: Scherzo

Arad, clasele a XI-a și a XII-a

O partitură muzicală este scrisă pentru V voci, fiecare voce având câte N note. Spunem că un solfegiu ***se brodește*** peste această partitură dacă solfegiul are lungimea de N note și fiecare din notele solfegiului coincide cu nota de pe poziția corespunzătoare a partiturii, pe cel puțin una dintre voci.

De exemplu, solfegiul `MI RE LA DO SI FA SOL LA` se brodește următoarei partituri pe două voci care conține opt note:

Vocea I: `MI RE DO DO SI FA FA SI`

Vocea II: `SOL SI LA DO SOL MI SOL LA`

Notele `1`, `2`, `5` și `6` din solfegiu corespund primei voci, iar notele `3`, `7`, `8` corespund celei de-a doua voci. A patra notă (`DO`) corespunde ambelor voci.

Spunem că un solfegiu de o lungime oarecare ***falsează*** în K locuri dacă este nevoie de cel puțin K modificări asupra lui pentru a-l face să se brodească peste partitură. Modificările admise sunt:

- ștergerea unei note din solfegiu;
- adăugarea unei note în solfegiu;
- înlocuirea unei note in solfegiu.

De exemplu, solfegiul `MI RE SOL DO SI FA SI` falsează în două locuri, pentru că trebuie să modificăm nota `SOL` în `LA` și să inserăm încă un `FA` între `SI` și `FA` pentru a-l aduce la forma `MI RE LA DO SI FA FA SI` care se brodește peste partitură.

Dumneavoastră sunteți profesor la conservator și, la examen, puneți studenții să reproducă o partitură la prima audiție, după care îi notați în funcție de cât de fidelă este reproducerea.

Evident, pentru a vă ușura munca, aveți nevoie de un program care să compare solfegiul (reproducerea) cu partitura și să spună dacă solfegiul se brodește sau în câte locuri falsează. În lumea prozaică a calculatoarelor, notele sunt reprezentate prin numere cuprinse între `1` și `100`.

Fișierul de intrare **SOLFEGIU.IN** conține pe prima linie numărul V ($1 \le V \le 100$) al vocilor și numărul N ($1 \le N \le 100$) al notelor, separate printr-un spațiu. Următoarele V linii conțin câte N numere care reprezintă notele fiecărei voci. Următoarea linie conține numărul M ($1 \le M \le 100$) al notelor din solfegiu, iar următoarea conține M numere întregi care reprezintă notele solfegiului.

Datele de ieșire se vor scrie în fișierul **SOLFEGIU.OUT** care va conține numărul minim de locuri în care falsează solfegiul. În cazul în care solfegiul se bordește peste partitură, în fișier se va scrie valoarea `0`.

Timp de execuție: 1 secundă/test

Exemplu

SOLFEGIU.IN	SOLFEGIU.OUT
2 8	2
3 2 1 1 7 4 4 7	
5 7 6 1 5 3 5 6	
7	
3 2 5 1 7 4 7	

P040129: Foaie de calcul

Satu Mare, clasele a XI-a și a XII-a

O foaie de calcul (*spreadsheet*) este un tablou dreptunghiular de celule rectangulare. Fiecare celulă poate conține date sau expresii, care pot fi evaluate obținându-se date. Liniile din foaia de calcul sunt numerotate, începând cu `0`, iar coloanele se denumesc cu litere majuscule ale alfabetului englez (primele `26` de linii), apoi folosind combinații de litere (`AA`, `AB` etc). O celulă poate fi referită specificând coloana și linia corespunzătoare (de exemplu, prima celulă este `A0`).

O foaie de calcul de jucărie conține cel mult `26` de coloane (denumite de la `A` la `Z`) și cel mult `10` linii (numerotate de la `0` la `9`). Fiecare celulă poate conține date de tip întreg sau expresii aritmetice în care se folosesc doar operatorii binari `+`, `-`, `*`, `/`, cu semnificația de adunare, scădere, înmulțire, respectiv împărțire întreagă. Operanzii pot fi valori întregi sau referințe către celule.

Problema constă în a evalua, dacă este posibil, o foaie de calcul de jucărie dată. Prin evaluarea unei foi de calcul înțelegem înlocuirea tuturor expresiilor cu valorile lor.

Pe prima linie a fișierului de intrare **EXCEL.IN** sunt scrise două numere întregi n (reprezentând numărul de

linii din foaia de calcul) și m (reprezentând numărul de coloane), separate prin spațiu.

Fișierul conține în continuare $n \times m$ linii de date, câte una pentru fiecare celulă. Conținutul celulelor (valoarea sau expresia) este specificat în ordinea crescătoare a liniilor, iar pe fiecare linie în ordinea alfabetică a coloanelor.

Fișierul de ieșire **EXCEL.OUT** va conține mesajul `EVALUARE IMPOSIBILA` sau foaia de calcul evaluată (n linii în ordinea dată, fiecare linie conținând cele m valori ale celulelor componente, în ordinea dată a coloanelor, separate prin spațiu).

Observații

- Expresiile pot conține paranteze rotunde.
- Expresiile nu conțin spații.
- Prioritatea operatorilor este cea uzuală.
- Datele și valorile absolute ale expresiilor din foaia de calcul de jucărie sunt numere naturale mai mici decât două miliarde.
- Fiecare expresie conține cel mult `100` de caractere.
- Expresiile date în foaia de calcul de jucărie sunt sintactic corecte (din punctul de vedere al sintaxei *Pascal/C*).

Timp de execuție: 0.5 secunde/test

Exemple

EXCEL.IN
```
1 2
1+B0
2+A0
```

EXCEL.OUT
```
EVALUARE IMPOSIBILA
```

EXCEL.IN
```
2 4
1
6
1
5
18
A1+A0
5
B0*3
```

EXCEL.OUT
```
1 6 1 5
18 19 5 18
```

P040130: Plimbare

Satu Mare, clasele a XI-a și a XII-a

Doi îndrăgostiți se află într-un oraș k și au la dispoziție p ore ($1 \le p \le 150$), timp în care să se plimbe cu mașina și apoi să ajungă fiecare la el acasă, fata în orașul i, iar băiatul în orașul j.

Ei doresc să se plimbe cât mai mult timp împreună, dar să ajungă în cel mult p ore, fiecare în orașul său. Mașina se deplasează cu viteză constantă (cea legal admisă).

Cei doi au la dispoziție o hartă cu n orașe ($3 \le V \le 200$) printre care se află, evident, orașul de plecare k și orașele destinație i și j. Harta indică orașele legate prin șosele și durata exprimată în ore a parcurgerii cu viteză constantă (legal admisă) a distanței dintre două orașe. Să se determine traseul comun pe care îl vor parcurge cei doi îndrăgostiți astfel încât timpul petrecut împreună să fie maxim și să ajungă amândoi la propria destinație în cel mult p ore de la plecare.

Observații

- Cronometrarea duratei plimbării se face începând cu momentul `0` din orașul de plecare k.
- În orice oraș unul dintre ei poate continua drumul separat, cu o altă mașină; (despărțirea nu necesită timp suplimentar).
- Pe fiecare șosea se circulă în ambele sensuri.
- După plecarea din orașul k nu se staționează nici un moment.
- Nu sunt permise întoarceri pe șosea sau parcurgeri incomplete ale unei șosele.

Pe prima linie a fișierului de intrare **MASINA.IN** se află două numere naturale n și m despărțite printr-un spațiu, reprezentând numărul de orașe de pe hartă și numărul de șosele ce leagă aceste orașe. Pe cea de a doua linie se află două numere naturale k și p, despărțite printr-un spațiu, reprezentând numărul de ordine al orașului de plecare și numărul de ore pe care îl au la dispoziție cei doi îndrăgostiți. Pe cea de a treia linie sunt scrise două numere naturale i și j despărțite printr-un spațiu, reprezentând numerele de ordine ale orașelor de destinație pentru cei doi. Pe următoarele m linii sunt scrise triplete de numere naturale a, b și d, despărțite prin câte un spațiu, reprezentând faptul că între orașele a și b există o șosea directă cu durata d de parcurgere a acesteia.

Rezultatele se vor scrie în fișierul **MASINA.OUT** structurate pe două linii. Pe prima linie este scris numărul natural *dmax*, reprezentând durata maximă (în ore) în care cei doi se vor plimba împreună. Pe linia a doua sunt scrise numerele naturale: $k, x_1, x_2, \ldots$ reprezentând orașele pe care le vor vizita împreună, în ordine, începând cu orașul k de plecare (datele vor fi despărțite prin câte un spațiu). În cazul în care soluția optimă nu este unică se va afișa o singură soluție.

Timp de execuție: 4 secunde/test

Exemplu

MASINA.IN
```
8 9
7 8
1 2
1 3 1
3 4 1
4 2 1
4 5 1
4 6 2
5 6 3
6 8 1
7 8 1
7 6 1
```

MASINA.OUT
```
6
7 6 5 4 3
```

P040131: Baterii

Suceava, clasele a XI-a și a XII-a

Direcția de informații a aflat prin agenții săi punctele de amplasare ale celor p ($1 \le p \le 10000$) baterii de apărare antiaeriană inamice, dispuse într-un sistem ortogonal de coordonate xOy.

Un atac fulger poate distruge tot ce se află într-un dreptunghi de dimensiuni $m \times n$.

Să se determine poziția unui dreptunghi cu laturile paralele cu axele de coordonate, de dimensiuni $m \times n$ ($1 \le m, n \le 10000$) date, care conține în interior sau pe frontieră un număr maxim de baterii inamice.

Fișierul de intrare **BATERII.IN** conține pe prima linie dimensiunile m și n ale dreptunghiului, pe a doua linie numărul total p al bateriilor inamice și pe următoarele p linii coordonatele celor p baterii inamice.

Coordonatele bateriilor sunt numere întregi cuprinse între 0 și 255.

Fișierul de ieșire **BATERII.OUT** va conține pe prima linie coordomatele u și v ale colțului din stânga-jos al dreptunghiului, pe a doua linie numărul k al bateriilor inamice distruse și pe următoarele k linii coordonatele celor k baterii inamice distruse.

Dacă există mai multe soluții în fișierul de ieșire se va scrie una singură.

Timp de execuție: 4 secunde/test

Exemplu

BATERII.IN	BATERII.OUT
3 2	2 2
5	4
3 3	2 3
4 2	3 3
2 3	4 2
5 4	5 4
1 5	

P040132: Segmente disjuncte

Suceava, clasele a XI-a și a XII-a

Se dau n ($1 \le n \le 1000$) segmente situate pe o dreaptă. Să se determine cardinalul maxim al unei mulțimi de segmente disjuncte de pe acea dreaptă.

Fișierul de intrare **SEGMENTE.IN** conține pe prima linie numărul n al segmentelor și pe următoarele n linii coordonatele celor n segmente, ordonate în funcție de coordonatele capătului din stânga.

Coordonatele segmentelor sunt numere întregi cuprinse între 0 și 255.

Fișierul de ieșire **SEGMENTE.OUT** va conține pe prima linie numărul maxim k al segmentelor disjuncte și, pe următoarea linie k numere întregi reprezentând numerele de ordine ale segmentelor rămase în mulțimea de cardinal maxim.

Timp de execuție: 1 secundă/test

Exemplu

SEGMENTE.IN	SEGMENTE.OUT
4	3
1 3	1 3 4
2 7	
4 5	
6 8	

P040133: Poștașul

Timiș, clasele a XI-a și a XII-a

Se dă un cartier ale cărui blocuri sunt poziționate sub forma unei matrice de patru linii și N coloane.

Pentru un N dat să se determine numărul variantelor de traseu pe care le poate parcurge poștașul zonal, astfel încât să treacă pe la toate blocurile, o singură dată pe la fiecare.

Poștașul pleacă de la blocul de pe poziția (1, 1) spre blocul de pe poziția (1, 2) și trebuie să se întoarcă la blocul (1, 1).

Valoarea N ($3 \le N \le 25$) se citește de la tastatură și rezultatul se afișează pe ecran.

Exemple

Intrare (tastatură)

```
3
```

Ieșire (ecran)

```
2
```

Intrare (tastatură)

```
17
```

Ieșire (ecran)

```
1029864
```

P040134: Multiplu

Timiș, clasele a XI-a și a XII-a

Scrieți un program care pentru un număr natural N ($3 < N \le 4999$) și pentru M cifre date $X_1, X_2, ..., X_M$ ($0 < M \le 9$, $0 \le X_i \le 9$) să găsească cel mai mic multiplu strict pozitiv al lui N care conține doar cifrele $X_1, X_2, ..., X_M$ (cifrele $X_1, X_2, ..., X_M$ pot apărea în multiplu niciodată, o dată sau de mai multe ori).

Numele fișierului de intrare și al celui de ieșire se vor citi de la tastatură.

Fișierul de intrare conține mai multe seturi de date separate de câte o linie vidă. Fiecare set de date are următoarea formă:

- pe prima line numărul N;
- pe a doua line numărul M;
- pe următoarele M linii cifrele $X_1, X_2, ..., X_M$.

Pentru fiecare set de date programul trebuie să tipărească în fișierul de ieșire, pe câte o linie, multiplul găsit, în cazul în care acesta există.

Dacă nu există soluție în fișierul de ieșire se va afișa doar valoarea 0.

Exemple

Intrare	*Ieșire*
3	110
22	0
3	20000999
7	
0	
1	
2	
1	
1	
4999	
6	
5	
7	
6	
2	
9	
0	

P040135: Printre isteți

Olt, clasele a XI-a și a XII-a

Verde Împărat a decis să-și căsătorească fiica. La curtea palatului sosesc mai mulți cavaleri care doresc să ceară mâna fetei.

Împăratul dorește să-l aleagă ca ginere pe cel mai isteț dintre ei. Pentru aceasta le cere să găsească soluția unei probleme pentru a cărei rezolvare se chinuie de mai mult timp. Se dă o matrice cu m linii și n coloane și se cere să se precizeze în câte moduri putem așeza valori `-1` și `1` astfel încât să se obțină pe fiecare linie și coloană produsul `-1`, dacă acest lucru este posibil. Cavalerii vă cer să-i ajutați la rezolvarea problemei.

Datele de intrare se citesc din fișierul **`ISTET.IN`** care conține dimensiunile m și n ($1 \le m, n \le 250$) ale rețelei, separate printr-un singur spațiu.

Datele de ieșire se vor scrie în fișierul **`ISTET.OUT`** care va conține numărul de posibilități de amplasare a valorilor `1` și `-1` astfel încât să se obțină pe fiecare linie și coloană produsul `-1`. Dacă nu există nici un astfel de amplasament se va scrie valoarea `0`.

Timp de execuție: 2 secunde/test

Exemplu

`ISTET.IN`	**`ISTET.OUT`**
3 3	16

P040136: Cărți

Olt, clasele a XI-a și a XII-a

N elevi au decis să schimbe cărți între ei, respectând algoritmul următor: fiecare elev dă cărți la jumătate dintre elevii cunoscuți de el și primește cărți de la cealalată jumătate de elevi care-l cunosc.

Se știe că fiecare elev are un număr par de cunoștințe și, un elev care a primit o carte de la un prieten nu poate să dea o carte la același elev.

Fiecare elev cunoaște cel puțin doi alți elevi.

Dacă elevul i îl cunoaște pe elevul j, atunci elevul j îl cunoaște și el pe elevul i.

Să se stabilească, pentru fiecare elev, care sunt elevii cărora trebuie să le dea cărți.

Datele de intrare se citesc din fișierul **`CARTI.IN`**. Pe prima linie se află numărul n al elevilor ($3 \le n \le 50$).

Pe următoarele n linii care conțin valori, separate între ele prin spații, reprezentând elevii cunoscuți de fiecare elev.

Pe prima linie se află elevii cunoscuți de primul elev, pe a doua elevii cunoscuți de al doilea ș.a.m.d.

Datele de ieșire vor fi scrise în fișierul **`CARTI.OUT`**. Pentru fiecare elev vor fi scrise câte două linii. Prima dintre ele va conține numărul c al elevilor care primesc cărți de la elevul respectiv, iar pe a doua vor fi scrise c valori, separate prin spații, care identifică cei c elevi care primesc cărți de la elevul respectiv.

Timp de execuție: 2 secunde/test

Exemplu

`CARTI.IN`

```
2
2 3 4 5
1 3 4 5
1 2 4 5
1 2 3 5
1 2 3 4
```

`CARTI.OUT`

```
5
2 4
2
3 5
2
1 4
2
2 5
2
1 3
```

Notă

Anumite enunțuri și exemple au fost modificate pentru a elimina unele neclarități.

ONI 2001

Mult succes tuturor participanților la Olimpiada Națională de Informatică!!!

CONCURSUL de programare AGORA

Cea de-a cincia rundă a Concursului de Programare Agora s-a încheiat. În acest număr vă vom prezenta soluțiile oficiale ale celor trei probleme, precum și rezultatele obținute de concurenți la această etapă. De asemenea, veți putea vedea care este clasamentul intermediar, după cinci etape.

Cele trei probleme ale rundei a patra au fost foarte simple dovadă fiind faptul că 21 dintre participanți au reușit să obțină toate cele 300 de puncte puse în joc.

Probleme cunoscute

Două dintre cele trei probleme propuse spre rezolvare erau destul de cunoscute, soluțiile lor putând fi găsite în diferite cărți care conțin probleme de informatică. La prima dintre ele majoritatea participanților au obținut punctajul maxim, dar la a doua numărul celor care au rezolvat corect s-a redus la aproape jumătate.

Cea de-a treia problemă se reducea la determinarea unui drum minim într-un graf. Chiar dacă, în cazul acestei probleme, costurile muchiilor se înmulțeau în loc să se adune, putea fi folosit un simplu artificiu matematic pentru a transforma operațiile de înmulțire în operații de adunare. Ca de obicei, operațiile cu numere mari au pus probleme multor concurenți; un număr mare de participanți nu au reușit să implementeze corect simularea operației de înmulțire.

Numărul de participanți

Datorită faptului că se apropie *Olimpiada Națională de Informatică*, numărul celor care au profitat de *Concursul de Programare Agora* pentru a-și testa cunoștințele a crescut semnificativ. Acest lucru nu poate decât să ne bucure deoarece scopul declarat al concursului nostru este de a vă ajuta să vă pregătiți pentru astfel de competiții.

Clasamentul rundei

1. Csaba Andras, Oradea	300	1. Vlad Vâlceanu, Arad	300	37. Andrei Csibi, Cluj-Napoca	203
1. Mugurel Ionuț Andreica, București	300	22. Andrei Giurgiu, București	294	37. Mihai Hărănguș, Cluj-Napoca	203
1. Ciprian Baicu, Dr. Tr. Severin	300	23. Horia Ciurdar, Timișoara	293	37. Codruț-Lucian Lazăr, Câmpeni	203
1. Adrian Balasko, Zalău	300	24. Arcadie Crăcan, Iași	290	37. Gabriel Pârvan, Rm. Sărat	203
1. Adrian Cârcu, Bistrița	300	25. Ciprian Cană, Vaslui	288	37. Lucian-Raul Silistru, Vaslui	203
1. George Drumea, București	300	26. Cristian Toth, Lugoj	280	37. Kristo Tammeoja, Estonia	203
1. Daniel Dumitran, București	300	27. Alexandra Constantin, Ploiești	260	47. Cristian Alexandrescu, Botoșani	200
1. Bogdan Dumitru, Ploiești	300	28. Radu Vătavu, Suceava	254	47. Elvis Asaftei, Vaslui	200
1. Dan Ghinea, București	300	29. Liviu-Cosmin Andreicuț, București	253	47. Constantin Asofiei, Tg. Neamț	200
1. Artur Kuczapski, Oradea	300	30. Valentin Pop, Carei	243	47. Andrei Benea, Focșani	200
1. Liviu Lalescu, Craiova	300	31. Vencel Bors, Oradea	229	47. Denis Bogdănaș, Iași	200
1. Alexandru Mosoi, Bacău	300	31. Maximilian Machedon, București	229	47. Cristian Bojinovici, București	200
1. Cosmin-Silevstru Negruseri, Bistrița	300	33. Marius Andrei, București	228	47. Alexandru Ivan, București	200
1. Mihai-Vlad Pantiș, Cluj-Napoca	300	33. Valentin Bisa, Lugoj	228	47. Bogdan Lucaciu, Dr. Tr. Severin	200
1. Tiberius Pârcălabu, București	300	35. Vincent Groenhuis, Olanda	214	47. Sorin Otescu, Brașov	200
1. Mihai Pătrașcu, Craiova	300	36. Mihail-Cosmin Pițiș, Dr. Tr. Severin	213	47. Flaviu Pașca, Bistrița	200
1. Liviu Păunescu, Constanța	300	37. Victor Asavei, Dr. Tr. Severin	203	47. Arpad Perini, Sf. Gheorghe	200
1. Sorin Stoiana, Craiova	300	37. Livia Bana, Dr. Tr. Severin	203	47. Bogdan-Milovan Piloga, Timișoara	200
1. Mihai Stroe, București	300	37. Daniel Bolohan, Iași	203	47. Mihai Stoicescu, Boldești-Scăeni	200
1. Radu Ștefan, Brașov	300	37. Bogdan Chirilă, Botoșani	203	47. Victor Vernescu, Constanța	200

Alb și negru

Această problemă a fost rezolvată corect de către 87 dintre cei care au participat la a cincia rundă a *Concursului de Programare Agora*. Chiar dacă problema nu era foarte ușoară, rezolvarea ei putea fi găsită în numeroase culegeri de probleme.

Metoda de rezolvare poate fi găsită foarte ușor folosind inducția matematică. Pentru început trebuie observat faptul că prima mutare admisă este fie deplasarea unei bile albe cu o poziție spre stânga, fie deplasarea unei bile negre cu o poziție spre dreapta. Vom presupune acum că vom muta bila albă; soluția corespunzătoare mutării unei bile negre la început poate fi găsită efectuând mutările simetric - în locul efectuării unei mutări cu o bilă albă vom efectua mutarea corespunzătoare cu o bilă neagră și invers. Fiecărei alternative îi corespunde o singură soluție.

Vom nota bilele albe cu `A` și bilele negre cu `B`. Configurația inițială este `|A...AAAA BBBB...B|`.

La primul pas vom muta bila albă; astfel vom ajunge în configurația `|A...AAA ABBBB...B|`.

Vom muta acum bila neagră pentru a ajunge în configurația `|A...AAABA BBB...B|`.

Urmează o nouă mutare cu o bilă neagră. Configurația în care se ajunge este `|A...AAABAB BB...B|`.

În acest moment vom efectua două mutări cu bilele albe. Configurația obținută este `|A...AA BABABB...B|`.

Urmează o deplasare cu o poziție a unei bile albe și apoi trei deplasări ale bilelor negre, cu două poziții. Configurația este `|A...ABABABA B...B|`.

Va urma o deplasare a unei bile negre cu o poziție și patru deplasări ale bilelor albe cu două poziții, o deplasare a unei bile albe cu o poziție și cinci deplasări ale bilelor negre cu două poziții etc.

După ce vom ajunge la pasul în care am efectuat n deplasări vom deplasa din nou o bilă cu o poziție și apoi vom efectua $n - 1$ deplasări cu două poziții etc. În final vom obține configurația cerută `|A...AAABA BBB...B|`.

Vom exemplifica algoritmul descris pentru $n = 4$:

Inițial: `|AAAA BBBB|`;
Pasul 1: `|AAA ABBBB|`;
Pasul 2: `|AAABA BBB|`;
Pasul 3: `|AAABAB BB|`;
Pasul 4: `|AAAB BABB|`;
Pasul 5: `|AA BABABB|`;
Pasul 6: `|A ABABABB|`;
Pasul 7: `|ABA ABABB|`;
Pasul 8: `|ABABA ABB|`;
Pasul 9: `|ABABABA B|`;
Pasul 10: `|ABABABAB |`;
Pasul 11: `|ABABAB BA|`;
Pasul 12: `|ABAB BABA|`;
Pasul 13: `|AB BABABA|`;
Pasul 14: `| BABABABA|`;
Pasul 15: `|B ABABABA|`;
Pasul 16: `|BBA ABABA|`;
Pasul 17: `|BBABA ABA|`;
Pasul 18: `|BBABABA A|`;
Pasul 19: `|BBABAB AA|`;
Pasul 20: `|BBAB BAAA|`;
Pasul 21: `|BB BABAAA|`;
Pasul 22: `|BBB ABAAA|`;
Pasul 23: `|BBBBA AAA|`;
Pasul 24: `|BBBB AAAA|`.

Implementarea este următoarea:

Listing BW.CPP

```
#include <stdio.h>
void main(void){
  int i,j,n;
  fscanf(fopen("BW.IN","rt"),"%d",&n);
  freopen("BW.OUT","wt",stdout);
  for (i=1;i<=n;i++){
    if (i%2){
      putchar('W');
      for (j=i;j--;putchar('b'));
    }
    else {
      putchar('B');
      for (j=i;j--;putchar('w'));
    }
  }
  for (i=n;i;i--){
    if (i%2){
      putchar('W');
      for (j=i;--j;putchar('w'));
    }
    else{
      putchar('B');
      for (j=i;--j;putchar('b'));
    }
  }
}
```

Au rezolvat corect:

Cristian Alexandrescu, Botoşani
Csaba Andras, Oradea
Marius Andrei, Bucureşti
Mugurel-Ionuţ Andreica, Bucureşti
Liviu-Cosmin Andreicuţ, Bucureşti
Elvis Asaftei, Vaslui
Victor Asavei, Drobeta Turnu Severin
Constantin Asofiei, Târgu Neamţ
Ciprian Baicu, Drobeta Turnu Severin
Adrian Balasko, Zalău
Radu Baluta, Târgu Jiu
Livia Bana, Drobeta Turnu Severin
Cristian Băicoianu, Ploieşti
Marius-Cosmin Belean, Arad
Andrei Benea, Focşani
Valentin Bişa, Lugoj
Denis Bogdănaş, Iaşi
Cristian Bojinovici, Bucureşti
Daniel Bolohan, Iaşi
Vencel Bors, Oradea
Ralph Bottesch, Sibiu
Ştefan Bucur, Bucureşti
Nadina Busuioc, Constanţa
Ciprian Cană, Vaslui
Adrian Cârcu, Bistriţa
Bogdan Chirilă, Botoşani
Horia Ciurdar, Timişoara
Radu Cocieru, Sibiu
Alexandra-Elena Constantin, Ploieşti
Horea Coroiu, Cluj-Napoca
Arcadie Crăcan, Iaşi
Andrei Csibi, Cluj-Napoca
Vlad Dascălu, Bacău
Andrei David, Râmnicu Vâlcea
Laurent Demonet, Franţa
George Drumea, Bucureşti
Octavian-Daniel Dumitran, Bucureşti
Bogdan Dumitru, Ploieşti
Aurel-Mihai Fulger, Constanţa
Dan Ghinea, Bucureşti
Andrei Giurgiu, Bucureşti
Mihai Gojol, Cluj-Napoca
Vincent Groenhuis, Olanda
Claudiu Gruia, Bucureşti
Mihai Hărănguş, Cluj-Napoca
Andrei Huţu, Bacău
Alexandru Ivan, Bucureşti
Silviu Julean, Arad
Artur Kuczapski, Oradea
Liviu Lalescu, Craiova
Codruţ-Lucian Lazăr, Câmpeni
Florin Leu, Mediaş
Bogdan Lucaciu, Drobeta Turnu Severin
Maximilian Machedon Bucureşti
Mihail-Cosmin Piţ-Rada, Drobeta Turnu Severin
Alexandru Mosoi, Bacău
Cosmin-Silvestru Negruseri, Bistriţa
Bogdan Nicolae, Sibiu
Sorin Otescu, Braşov
Mihai-Vlad Pantiş, Cluj-Napoca
Tiberius Pârcălabu, Bucureşti
Gabriel Pârvan, Râmnicu Sărat
Flaviu Paşca, Bistriţa
Csaba Pătcaş, Oradea
Mihai Pătraşcu, Craiova
Liviu Păunescu, Constanţa
Arpad Perini, Sfântu Gheorghe
Bogdan-Milovan Piloga, Timişoara
Valentin Pop, Carei
Mihail Popa, Bucureşti
Dan Popovici, Arad
Cosmin Răianu, Constanţa
Dan-Octavian Savu, Bucureşti
Lucian-Raul Silistru, Vaslui
Johannes Slotta, Germania
Bogdan Stan, Câmpina
Lucian-Daniel Stanciu, Brăila
Sorin Stoiana, Craiova
Mihai Stoicescu, Boldeşti-Scăeni
Mihai Stroe, Bucureşti
Radu Ştefan, Braşov
Kristo Tammeoja, Estonia
Cristian Toth, Lugoj
Silviu-Gabriel Udrea, Târgu Jiu
Vlad Vâlceanu, Arad
Radu Vătavu, Suceava
Victor Vernescu, Constanţa

Runda 5, problema 2

Coeficient

Această problemă a fost rezolvată corect de către 23 dintre cei care au participat la a cincia rundă a *Concursului de Programare Agora*. Transpusă în termeni de teoria grafurilor problema cerea determinarea drumului minim între două vârfuri ale unui graf, dar costul unui drum nu era dat de suma costurilor muchiilor componente, ci de produsul lor.

Totuși, ea poate fi foarte ușor transformată în problema clasică a drumului minim dacă transformăm operația de înmulțire în operație de adunare. Acest lucru poate fi realizat foarte ușor dacă logaritmăm costurile muchiilor și determinăm logaritmul costului drumului minim, după care calculăm foarte simplu și costul real al acestui drum.

Datorită limitei specificate de 1000 de noduri, era suficient să implementăm versiunea algoritmului lui *Dijkstra* cu complexitatea $O(n^2)$.

O variantă de implementare este următoarea:

Listing COEF.CPP

```
#include <stdio.h>
#include <math.h>
#include <alloc.h>

int n;
int len[1000];
int t[1000];
int sol[5000],
int far *links[1000];
int far *c[1000];
double far *clog[1000];
double d[1000];

void ReadData(void){
  int i,j,m,x,y,cost;
  double cost_log;
  FILE *f=fopen("COEF.IN","rt");
  fscanf(f,"%d%d",&n,&m);
  for (i=0;i<n;len[i++]=0);
  for (i=0;i<m;i++){
    fscanf(f,"%d%d%*d",&x,&y);
    len[--x]++;
    len[--y]++;
  }
  fclose(f);
```

Au rezolvat corect:

Csaba Andras, Oradea
Mugurel Ionuț Andreica, București
Ciprian Baicu, Dr. Tr. Severin
Adrian Balasko, Zalău
Vencel Bors, Oradea
Adrian Cârcu, Bistrița
Arcadie Crăcan, Iași
George Drumea, București
Daniel Dumitran, București
Bogdan Dumitru, Ploiești
Dan Ghinea, București
Artur Kuczapski, Oradea
Liviu Lalescu, Craiova
Alexandru Mosoi, Bacău
Silvestru Negruseri, Bistrița
Mihai-Vlad Pantiș, Cluj-Napoca
Tiberius Pârcălabu, București
Mihai Pătrașcu, Craiova
Liviu Păunescu, Constanța
Sorin Stoiana, Craiova
Mihai Stroe, București
Radu Ștefan, Brașov
Vlad Vâlceanu, Arad

```
  for (i=0;i<n;i++){
    links[i]=(int far*)
             farmalloc(len[i]*sizeof(int));
    c[i]=(int far*)
             farmalloc(len[i]*sizeof(int));
    clog[i]=(double far*)
          farmalloc(len[i]*sizeof(double));
    t[i]=0;
  }
  f=fopen("COEF.IN","rt");
  fscanf(f,"%*d%*d");
  for (i=0;i<m;i++){
    fscanf(f,"%d%d%d",&x,&y,&cost);
    j=t[--x];
    links[x][j]=--y;
    c[x][j]=cost;
    clog[x][j]=log(cost);
    t[x]++;
    j=t[y];
    links[y][j]=x;
    c[y][j]=cost;
    clog[y][j]=log(cost);
    t[y]++;
  }
  fclose(f);
}

void Multiply(unsigned long a){
  unsigned long k;
  int i;
  k=0;
  for (i=1;i<=sol[0];i++){
    k+=a*sol[i];
    sol[i]=k%100;
    k/=100;
  }
  while (k){
    sol[++sol[0]]=k%100;
    k/=100;
  }
}

void FindShortestPath(void){
  int i,j,imin,k;
  double min;
  char s[1000];
  for (i=0;i<n;i++){
    d[i]=1.0/0.0;
    s[i]=0;
    t[i]=-1;
  }
  for (i=0;i<len[0];i++){
    d[links[0][i]]=clog[0][i];
    t[links[0][i]]=0;
  }
  k=0;
  d[0]=0;
  s[0]=1;
  t[0]=-1;
  while (1){
    k++;
    imin=-1;
    min=1000000.0f;
    for (i=0;i<n;i++)
      if (!s[i] && d[i]<min){
        min=d[i];
        imin=i;
      }
    if (imin==-1)
      break;
    s[imin]=1;
    for (i=0;i<len[imin];i++){
      j=links[imin][i];
      if (!s[j] &&
              d[imin]+clog[imin][i]<d[j]){
        d[j]=d[imin]+clog[imin][i];
        t[j]=imin;
      }
    }
  }
}

void WriteSolution(){
  int i,j;
  FILE *f=fopen("COEF.OUT","wt");
  sol[0]=sol[1]=1;
  for (i=n-1;i;i=t[i]){
    for (j=0;j<len[i];j++)
      if (links[i][j]==t[i]){
        Multiply(c[i][j]);
        break;
      }
  }
  fprintf(f,"%d",sol[sol[0]]);
  for (i=sol[0]-1;i;i--)
    fprintf(f,"%02d",sol[i]);
  fprintf(f,"\n");
  fclose(f);
}

void main(void){
  ReadData();
  FindShortestPath();
  WriteSolution();
}
```

Notă

Această implementare îi aparține lui **George Drumea** *din București, concurentul clasat pe primul loc înaintea ultimei etape a* **Concursului de Programare Agora** *organizat de* **Gazeta de Informatică**.

Runda 5, problema 3

Matrice

Această problemă a fost rezolvată corect de 59 dintre cei care au participat la a cincia rundă a *Concursului de Programare Agora*. Problema a fost preluată din cartea *Proiectarea și implementarea algoritmilor*, scrisă de *Mihai Oltean*.

Este evident că efectuarea a două operații asupra aceleiași linii sau a aceleiași coloane nu are nici un efect.

O valoare -1 poate părea pe poziția (i, j) dacă s- efectuat o operație asupra liniei i și nu s-a efectuat o operație asupra linie j sau invers.

Dacă notăm cu x numărul operațiilor efectuate asupra liniilor și cu y al celor efectuate asupra coloanelor, atunci numărul total de valori -1 va fi $x \cdot (m - y) + y \cdot (n - x)$. Această valoare trebuie să fie k, deci va trebui să rezolvăm ecuația $x \cdot (m - y) + y \cdot (n - x) = k$.

Vom rezolva ecuația considerând toate perechiile (x, y) posibile. În cazul în care aceasta are soluție, putem alege oricare x linii și oricare y coloane.

Algoritmul descris poate fi implementat astfel:

Listing MATRIX.CPP

```
#include <stdio.h>
void main(void){
  long x,y,m,n,k;
  FILE *f=fopen("MATRIX.IN","rt");
  fscanf(f,"%ld%ld%ld",&m,&n,&k);
  fclose(f);
  f=fopen("matrix.out","wt");
  for (x=0;x<=m;x++)
    for (y=0;y<=n;y++)
      if (x*(n-y)+y*(m-x)==k){
        while (x) fprintf(f,"L%d\n",x--);
        while (y) fprintf(f,"C%d\n",y--);
        fclose(f); return;
      }
  fprintf(f,"0\n");
  fclose(f);
}
```

Au rezolvat corect:

Cristian Alexandrescu, Botoşani
Csaba Andras, Oradea
Marius Andrei, Bucureşti
Mugurel Andreica, Bucureşti
Elvis Asaftei, Vaslui
Victor Asavei, Dr. Tr. Severin
Constantin Asofiei, Târgu Neamț
Ciprian Baicu, Dr. Tr. Severin
Adrian Balasko, Zalău
Livia Bana, Dr. Tr. Severin
Andrei Benea, Focşani
Valentin Bisa, Lugoj
Denis Bogdănaş, Iaşi
Cristian Bojinovici, Bucureşti
Daniel Bolohan, Iaşi
Ciprian Cană, Vaslui
Adrian Cârcu, Bistrița
Bogdan Chirilă, Botoşani
Horia Ciurdar, Timişoara
Alexandra Constantin, Ploieşti
Andrei Csibi, Cluj-Napoca
George Drumea, Bucureşti
Daniel Dumitran, Bucureşti
Bogdan Dumitru, Ploieşti
Dan Ghinea, Bucureşti
Andrei Giurgiu, Bucureşti
Vincent Groenhuis, Olanda
Mihai Hărănguş, Cluj-Napoca
Alexandru Ivan, Bucureşti
Artur Kuczapski, Oradea
Liviu Lalescu, Craiova
Lucian Lazăr, Câmpeni
Bogdan Lucaciu, Dr. Tr. Severin
Maximilian Machedon, Bucureşti
Mihail-Cosmin Piț-Rada, Dr. Tr. Severin
Alexandru Mosoi, Bacău
Silvestru Negruseri, Bistrița
Sorin Otescu, Braşov
Mihai Pantiş, Cluj-Napoca
Tiberius Pârcălabu, Bucureşti
Gabriel Pârvan, Râmnicu Sărat
Flaviu Paşca, Bistrița
Mihai Pătraşcu, Craiova
Liviu Păunescu, Constanța
Arpad Perini, Sfântu Gheorghe
Bogdan Piloga, Timişoara
Valentin Pop, Carei
Mohammad Ali Safari Ghahsareh, Iran
Lucian-Raul Silistru, Vaslui
Sorin Stoiana, Craiova
Mihai Stoicescu, Boldeşti-Scăeni
Mihai Stroe, Bucureşti
Radu Ştefan, Braşov
Kristo Tammeoja, Estonia
Cristian Toth, Lugoj
Vlad Vâlceanu, Arad
Radu Vătavu, Suceava
Victor Vernescu, Constanța
Violeta Voinescu, Târgu Jiu

GInfo Top 100

1. George Drumea, Bucure ti	1465	6. Maximilian Machedon, Bucure ti	1400
2. Ciprian Can Vaslui	1449	7. Csaba Andras, Oradea	1399
3. Radu tefan, Bra ov	1424	8. Mihai Stroe, Bucur ti	1389
4. Adrian Cârcu, Bistri	1414	9. Mugurel Ionu Andreica, Bucure ti	1329
5. Silvestru Negruseri, Bistria	1409	10. Liviu Ialescu, Craiova	1328

11. Dan Ghinea Bucure ti	1313	41. Andrei Csibi, Cluj-Napoca	976	71. Andrei Markovits, Satu Mare	546
12. Ciprian Baicu, Dr. Turnu Severin	1306	42. Valentin Bisa, Lugoj	947	72. Laurent Demonet, Fran	520
13. Cristian B coianu, Ploieti	1256	43. Vlad Dascu, Bac	935	73. Lucian-Daniel Stanciu, B la	518
14. Mihai P tra cu, Craiova	1246	44. Mihai H ngu , Cluj-Napoca	921	74. tefan Bucur, Bucure ti	478
15. Gabriel Pârvan, Râmnicu Sat	1237	45. Liviu-Cosmin Andreicu Bucure ti	893	75. Daniel Bolohan, Ia	469
16. Vlad Vâlceanu, Arad	1218	46. Artur Kuczapski, Oradea	891	76. Aurel-Mihai Fulger, Constan	460
17. Liviu P nescu, Constana	1216	47. Marius Andrei, Bucure ti	886	77. Constantin Asofiei, Târgu Neam	451
18. Octavian-Daniel Dumitran, Bucure ti	1214	48. Andrei David, Râmnicu Vâlcea	872	78. Raluca Mu lotu-Elefteri, Bucure ti	443
19. Denis Bogd a , Ia i	1204	49. Andrei Giurgiu, Bucure ti	864	79. Bogdan Chiril Boto ani	440
20. Cristian Alexandrescu, Botani	1176	50. Codru-Lucian Laz , Câmpeni	861	80. Mihai Gojol, Cluj-Napoca	438
21. Bogdan Stan, Câmpina	1165	51. Victor Vernescu, Constan	855	81. Petru-Simon Mo Br la	437
22. Claudiu Gruia, Bucure ti	1155	52. Horea Coroiu, Cluj-Napoca	850	82. Mihai Stoicescu, Boldei-Sc ni	431
23. Victor Asavei, Drobeta Turnu Severin	1147	53. Ioana-Maria Ilean Alba Iulia	824	83. V. Muthuramakrishnan, India	421
24. Mihai-Vlad Pantiu Cluj-Napoca	1141	54. Valentin Pop, Carei	820	84. Nadina Busuioc, Constan	415
25. Vencel Bors, Oradea	1129	55. Aurelian Ghi , Buz	785	85. Andrei-Liviu Dumbrava, Ia	414
26. Bogdan Dumitru, Ploieti	1111	56. M.C. Pi-Rada, Dr. Turnu Severin	758	86. Tiberius Pârcalabu, Bucureti	412
27. Radu V tavu, Suceava	1092	57. Lucian-Raul Silistru, Vaslui	757	87. Silviu-Gabriel Udrea, Târgu Jiu	408
28. Alexandra-Elena Constantin, Ploie	1068	58. Arcadie Cr an, Ia i	746	88. Elvis Asaftei, Vaslui	400
29. Horia Ciurdar, Timoara	1056	59. Cosmin R anu, Constana	745	89. Johannes Slotta, Germania	398
29. Vincent Groenhuis, Olanda	1056	60. R zvan Mu lotu-Elefteri, Bucure ti	706	90. Rusu Paul, Cluj-Napoca	396
31. Alexandru Mosoi, Bac	1055	61. C lin Kesa, Buteni	696	91. Radu Cocieru, Sibiu	367
32. Mihail Popa, Bucure ti	1038	62. Paul Marinescu, Buz	693	92. Petko Minkov, Bulgaria	366
33. Cristian Toth, Lugoj	1034	63. Bogdan-Milovan Piloga, Timiara	689	93. Bogdan-Vlad ira, Cluj-Napoca	349
34. M.A. Safari Ghahsareh, Iran	1020	64. Radu Dondera, Bucure ti	617	94. Emilian Miron, Bac	337
35. Daniel Crian, Râmnicu Vâlcea	1015	65. Bogdan Lucaciu, Dr. Turnu Severin	612	95. Raluca Sauciuc, Bucure ti	330
36. Dan Popovici, Arad	1013	66. Adrian Balasko, Zal	604	96. Lucian Burja Blaj,	320
37. Flaviu Paca, Bistria	1011	67. Andrei Benea, Foani	576	97. Costin Speciac Buz,	316
38. Bogdan Nicolae, Sibiu	1007	68. Adrian Cearn , Bucureti	569	98. Arpad Perini, Sfântu Gheorghe	312
39. Csaba P tra , Oradea	1004	69. Livia Bana, Drobeta Turnu Severin	562	99. Dan-Octavian Savu, Bucure ti	305
40. Sorin Otescu, Bra ov	984	70. Cristian Bojinovici, Bucure ti	553	100. Sorin Stoiana, Craiova	300

O problemă de COMBINATORICĂ

Tiberiu Socaciu

Dedic acest articol domnului Mihail Hudrea, profesor la Liceul de Informatică "Tiberiu Popoviciu" din Cluj, pentru că nu pot uita orele de ASSIRIS (limbajul de asamblare al calculatoarelor FILIX 256) pe care le ținea cu noi acum 15 ani (pe atunci eram elev al Liceului de Matematică-Fizică Nr. 2 - Informatică din Cluj). Chiar dacă domnia sa se ocupă acum mai mult de matematică, sper că va citi acest articol cu plăcere.

O problemă de numărare

Enunțul acestei probleme de combinatorică l-am primit de la *dl. Mihail Hudrea*: "*Fie n un număr natural. Notăm cu S_n mulțimea tuturor numerelor formate numai cu cifrele 1 și 3, astfel încât suma cifrelor să fie n. Să se afle numărul de elemente ale mulțimii S_n.*"

Rezolvarea problemei

Vom încerca să scriem un număr din S_n folosind un număr de a cifre de 1 și b cifre de 3; deoarece suma cifrelor este n, înseamnă că a și b verifică ***ecuația diofantică*** (adică o ecuație ai cărei coeficienți și ale cărei soluții sunt numere întregi) $a + 3b = n$, cu condițiile evidente $a \geq 0$ și $b \geq 0$, adică $a = n - 3b$, unde b ia valori succesive între 0 și $[n / 3]$, unde $[x]$ reprezintă partea întreagă a numărului real x. Pentru o soluție (a, b) a ecuației diofantice există un număr de $C_{a+b}^{a} = C_{a+b}^{b}$ elemente distincte ale mulțimii S_n care respectă condiția menționată. Prin urmare, numărul de elemente ale lui S_n este $C_n^0 + C_{n-2}^1 + C_{n-4}^2 + \ldots + C_{n-2\cdot[n/3]}^{[n/3]}$, unde suma are $[n / 3] + 1$ termeni.

Enunțul original

Enunțul original cerea să se arate că în anumite condiții impuse asupra valorii n, numărul de elemente ale mulțimii S_n este un pătrat perfect. Rezolvarea se bazează pe scrierea unor relații de recurență de forma $X_n - X_{n-1} = X_{n-3}$, unde cu X_n am notat numărul de elemente ale mulțimii S_n.

Generalizarea problemei

O primă generalizare este imediată: "*Fie n un număr natural și a o cifră diferită de 1 și 0. Notăm cu S_n mulțimea tuturor numerelor formate numai cu cifrele 1 și a, astfel încât suma cifrelor să fie n. Să se afle numărul de elemente ale mulțimii S_n.*"

Rezolvarea problemei generalizate

Procedând analog, obținem că numărul de elemente ale mulțimii S_n este $C_n^0 + C_{n-(a-1)}^1 + C_{n-2\cdot(a-1)}^2 + \ldots + C_{n-[n/a]\cdot(a-1)}^{[n/a]}$, unde suma are $[n / a] + 1$ termeni.

Evident, pentru $a = 3$ obținem problema inițială. Scăzând termenii consecutivi ai șirului X_n (definit ca în paragraful anterior) și aplicând regula lui *Pascal* pentru toți termenii, mai puțin primul (care se reduce deoarece are valoarea 1) obținem $X_n - X_{n-1} = X_{n-a}$ (trebuie remarcat faptul că pentru $n \equiv 0 \pmod a$ în X_n și X_{n-a} apare încă un termen egal cu 1). Evident, valorile de pornire sunt $X_i = 1$ pentru $i = 1, \ldots, a - 1$ și $X_a = 2$.

Implementarea

Pentru implementare am ales un șir X de dimensiune a în care reținem de fiecare dată numai ultimii a termeni ai șirului, adică exact termenii de care avem nevoie pentru a calcula termenul curent cu ajutorul recurenței. Pentru implementare a fost ales limbajul *Java* și a fost testat pe o mașină *Linux* cu *JDK 1.2* (vezi [*Java*]).

Listing CALCUL.JAVA:

```
public class calcul{
  private static int rezolva(int a,
                             int n){
    if ((n<=1)||(a<=1))
      return 0;
```

```
    int x[]=new int[a];
    x[0]=2;
    for(int i=1;i<a;i++)
      x[i]=1;
    for(int i=a+1;i<=n;i++)
      x[i%a]=x[(i-1)%a]+x[(i-a)%a];
    return x[n%a];
  }

  public static void main(String
                                    args[]){
    System.out.println(rezolva(3,20));
  }
}
```

O generalizare mai puternică

Încercăm acum o generalizare mai puternică, în sensul că vom lăsa pentru alegerea cifrelor o mai mare libertate.

"Fie n un număr natural și fie $M = \{a_1, ..., a_p\}$ o mulțime de p cifre într-o bază oarecare de numerație. Notăm cu S_n mulțimea tuturor numerelor formate numai cu cifre din M, astfel încât suma cifrelor să fie m. Să se afle numărul de elemente ale mulțimii S_n."

De remarcat este faptul că dacă $M = \{1, ..., n-1\}$, atunci problema este echivalentă cu determinarea numărului de partiții ale lui *n*.

Combinări binomiale și combinări multinomiale

Din definiția combinatorială a combinărilor rezultă imediat ca proprietate formula de dezvoltare binomială a lui *Newton*: $(a+b)^n = C_n^0 a^n + \text{L} + C_n^k a^{n-k} b^k + \text{L} + C_n^n b^k$, unde $C_n^k = \frac{n!}{k!(n-k)!}$. (Datorită proprietăților binomiale, acestea combinări se numesc și ***combinări binomiale***). Justificarea coeficienților formulei lui *Newton* se face pe baza numărării termenilor monomiali asemenea, obținuți în urma dezvoltării (vezi [*Manual*]).

Prin extindere, vom denumi ***combinare multinomială*** coeficientul din dezvoltarea unui multinom la o putere *n* dată. Vom nota cu $(n \mid k_1, ..., k_p)$ coeficientul termenului $x_1^{k_1} \text{L}\ x_p^{k_p}$ din dezvoltarea $(x_1 + ... + x_p)^n$. Acesta este egal cu $\frac{n!}{k_1! \text{L}\ k_p!}$.

Demonstrația este relativ simplă și se bazează pe metoda inducției matematice. Este evident că $k_1 + ... + k_p = n$, deoarece fiecare monom din dezvoltare are gradul *n*. Pentru $p = 2$, obținem combinările binomiale: $(n \mid k, n - k) = C_n^k = C_n^{n-k}$.

Pentru implementare am preferat să folosim definiția cu factoriali a combinărilor (vezi funcția `Combinare` din programul **CALCUL.PAS**).

Rezolvarea noii probleme generalizate

Încercăm să scriem un număr din S_n folosind x_1 cifre a_1, ..., x_p cifre a_p. Deoarece suma cifrelor este *n*, avem că x_i verifică ecuația diofantică $a_1x_1 + ... + a_px_p = n$, respectând condițiile evidente $x_i \geq 0$.

Pentru o soluție $(x_1, ..., x_p)$ a ecuației diofantice există un număr de $(x_1 + ... + x_p \mid x_1, ..., x_p)$ elemente distincte ale mulțimii S_n dacă se respectă condiția menționată.

Prin urmare, numărul de elemente ale mulțimii S_n este suma combinărilor multinomiale corespunzătoare pentru fiecare soluție a ecuației diofantice.

Backtracking și funcția de continuare pentru rezolvarea ecuației diofantice

Din cele expuse anterior, dificultatea o constituie rezolvarea ecuației diofantice $a_1x_1 + ... + a_px_p = n$ în numere naturale. Rezultă că, din punct de vedere tehnic, limitările pentru x_i sunt date de intervalul $[0, n / a_i]$.

Rezolvarea noastră se va baza pe metoda *backtracking*. Pentru aceasta, avem nevoie de o funcție de continuare cu proprietățile din [*Socaciu*].

O bună condiție de continuare poate fi obținută în mod curent din condiții de necesitate. Astfel, presupunând că $(x_1, ..., x_p)$ este o soluție, rezultă că se verifică ecuația diofantică și, din pozitivitatea soluțiilor, obținem că $a_1x_1 + ... + a_kx_k \leq n$ pentru orice $k = 1, p - 1$. Prin urmare, am obținut o funcție de continuare bună (vezi funcția `fi` din programul **CALCUL.PAS**).

Implementarea

Implementarea a fost realizată în *Pascal*, programul fiind compilat pe o platformă *Win32* cu compilatorul *FreePascal 0.99.14* distribuit sub licență *GNU* (vezi [*FreePascal*]).

Teoretic, ar trebui să reușiți compilarea acestui program fără probleme pe orice platformă cu acest compilator, dar și sub clasicele compilatoare *Turbo/Borland* pentru *DOS*.

Listing CALCUL.PAS

```
Program calcul;
     { datele problemei si rezultatul }
type sir=array[1..100] of Integer;
var rezultat,n,p:Integer;
    a: sir;

function combinare(p:Integer;x:sir):
                                  Integer;
{ calculeaza o combinare multinomiala }
{ de forma (x1+...+xp|x1,...,xp)      }
var c:Real;
    s,i:Integer;
begin
  if p=1 then
    if x[1]=0 then combinare:=0
    else combinare:=1
```

```
  else
    begin
    c:=combinare(p-1,x);
    s:=0;
    for i:=1 to p-1 do
      s:=s+x[i];
    for i:=1 to x[p] do
    begin
      Inc(s);
      c:=c*s/i
    end;
    combinare:=trunc(round(c))
  end
end;

{ solutia ecuatiei                    }
{     a1 * x1 + ... + ap * xp = n     }
{ in numere naturale                  }
var x:sir;

function fi(k:Integer):Boolean;
{ functia de continuare }
var i,s:Integer;
begin
  s:=0;
  for i:=1 to k do
    Inc(s,a[i]*x[i]);
  fi:=((k=p) and (s=n)) or
      ((k<p) and (s<=n))
end;

procedure rezolva(k:Integer);
{ determina solutiile ecuatiei }
{ folosind metoda backtracking }
var i:Integer;
begin
  if k=p+1 then
    Inc(rezultat,combinare(p,x))
  else
    for i := 0 to n div a[k] do
    begin
      x[k]:=i;
      if fi(k) then rezolva (k+1)
    end
end;

procedure citeste;
{ citeste datele }
var i:Integer;
begin
  Read(p);
  for i:=1 to p do
    Read(a[i]);
  Read(n)
end;
```

```
Begin
  rezultat:=0;
  citeste;
  rezolva(1);
  writeln(rezultat);
End.
```

Teme

Încercați-vă puterile cu următoarele probleme:

Problema 1

Modificați funcția `rezolva()` din cadrul programului **CALCUL.JAVA** prin eliminarea operatorului `%` (modulo) și folosirea operatorului `+=`.

Problema 2

Modificați cele două programe prezentate în cadrul acestui articol astfel încât să accepte datele de intrare dintr-un fișier după un anumit șablon.

Problema 3

Realizați o rutină de verificare a unicității elementelor a_i (elementele șirului a trebuie să fie distincte).

Problema 4

Găsiți o relație de recurență pentru combinările multinomiale (analoagă relației triunghiulare a lui *Pascal* de la combinările binomiale) și reimplementați optimal funcția `combinare`. (Eventual, puteți ține cont și de ordinea de apelare a acestei funcții - în versiunea prezentată se apelează în ordinea găsirii soluțiilor ecuației diofantice).

Problema 5

Modificați programul astel încât să se construiască efectiv toate elementele mulțimii S_n (eventual folosind metoda *backtracking*). Care dintre următoareale abordări vi se pare mai bună: imbricarea a două *backtracking*-uri (întâi cel "mare" care rezolvă ecuația, apoi cel "mic" care determină soluțiile de același tip) sau un singur *backtracking*? Argumentați varianta aleasă.

Problema 6

Demonstrați că numărul de elemente ale mulțimii S_n poate fi scris sub forma $k_1(m_1)^n + \ldots + k_q(m_q)^n$. Deoarece există posibilitatea ca cel puțin valoarea absolută a uneia dintre valorile m_i să fie subunitară, rezultă că sunt șanse ca numărul de elemente ale lui S_n să aibă o creștere exponențială, astfel tipul `Integer` să fie repede depășit. Modificați programul astfel încât acest risc să fie eliminat.

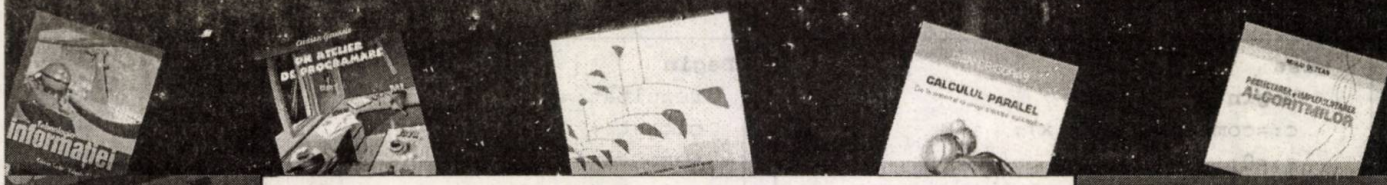
Bibliografie

[*FreePascal*] http://www.freepascal.org

[*Java*] http://www.java.sun.com

[*Manual*] Manuale de *Algebră* pentru liceu

[*Socaciu*] Tiberiu Socaciu, *Metoda Backtracking*, revista *if*, nr. 5(16)/1992, pag. 22-25.



COMPUTER LIBRIS AGORA

"Cum aș putea să justific existența acestei cărți? Cum aș putea spre exemplu să justfic cele 100 de pagini despre programarea dinamică, acum când această metodă nu mai figurează în programele școlare? Nu știu. Sau, cum aș putea să justific capitolul despre algoritmi genetici, când pe plan mondial abia de câțiva ani a fost dovedită utilitatea și eficiența lor? Din nou nu știu! Dar rămâne bănuiala: a scrie algoritmi înseamnă a crea. Și nu poți crea fără să gândești."

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Sisteme de FIȘIERE

Ciprian Ileană, Claudiu Soroiu

În cadrul acestui articol vom prezenta sistemul de fișiere folosit de sistemele de operare produse de Microsoft. Vom începe cu sistemul FAT 12 și vom continua cu FAT 16 și FAT 32.

Introducere

Sistemul de fișiere *FAT* (*File Allocation Table*) a apărut la sfârșitul anilor '70, fiind sistemul de fișiere suportat de sistemul de operare *MS-DOS*.

Inițial, a fost dezvoltat ca un simplu sistem de fișiere pentru *floppy disk*-uri cu dimensiune mai mică de 500 KB. Cu timpul a fost îmbunătățit, pentru a suporta medii mai mari de date. În prezent sunt trei tipuri de *FAT*: *FAT12*, *FAT16* și *FAT32*. Diferența de bază între aceste trei tipuri de *FAT* este aceea (de unde și numele) că intrările fișierelor din *FAT* au mărimi diferite ca număr de biți. Într-o intrare de *FAT12* sunt folosiți 12 biți, pentru *FAT16* sunt folosiți 16 biți iar pentru *FAT32* se folosesc 32 de biți.

Privire de ansamblu

La origine, toate sistemele de fișiere *FAT* au fost dezvoltate pentru PC-uri cu arhitecturi compatibile *IBM*. În consecință, toate structurile de date sunt "*little endian*" (datele sunt scrise în format ***LSB*** - ***L**ess **S**ignificant **B**yte **F**irst*). De exemplu, o intrare de *FAT32* este stocată pe disc ca o serie de patru bytes (1 byte = 8 biți), primul fiind byte[0] iar ultimul byte[3].

În cele ce urmează se observă numerotarea celor 32 de biți începând cu bitul 0 (0 este cel mai nesemnificativ bit - dreapta jos).

```
31 30 29 28 27 26 25 24 - byte-ul 3
23 22 21 20 19 18 17 16 - byte-ul 2
15 14 13 12 11 10 09 08 - byte-ul 1
07 06 05 04 03 02 01 00 - byte-ul 0
```

Este important dacă un anumit sistem este unul "*big endian*" (cum sunt sistemele livrate de *Motorola*), deoarece în timpul transferurilor de date intervine transformarea acestora din ***MSB*** (*Most Significant Byte First*) în ***LSB*** și invers.

Un sistem de fișiere *FAT* este compus din patru zone de bază, care sunt aranjate în ordinea care urmează:

- zona rezervată
- zona *FAT* (tabelele de alocare a fișierelor)
- zona directorului rădăcină (*ROOT* - a fost eliminată începând cu *FAT32*)
- zona de date a fișierelor și directoarelor (*FDDR*).

Zona Rezervată

Prima structură de date importantă pe un volum *FAT* este numită *BPB* (*BIOS Parameter Block*) și este localizată în primul sector al volumului (partiției), în zona rezervată. Acest sector este numit uneori "*boot sector*" (sector de *boot*) sau "*sectorul 0*", dar lucrul cel mai important este că acesta este primul sector al volumului (partiției).

Acesta este un prim element care cauzează uneori confuzie. În *MS-DOS* versiunea `1.x` nu exista *BPB* în sectorul de *boot*. În această primă versiune a sistemului de fișiere *FAT*, existau doar două formate diferite: unul pentru *floppy disk*-uri cu o singură față și unul pentru *floppy disk*-uri cu două fețe (de 360K - 5.25"). Determinarea tipului era realizată pe baza primului byte din *FAT* (primii 8 biți ai lui FAT[0]).

Acest mod de determinare a formatului a fost eliminat începând cu versiunea `2.x` prin adăugarea unui *BPB* în sectorul de *boot*. Astfel, vechiul stil de determinare a fost complet eliminat. În consecință toate volumele (partițiile) *FAT* trebuie să aibă un *BPB* în sectorul de boot.

Acest fapt duce la o nouă (posibilă) confuzie, relativ la determinarea tipului de *FAT*.

În continuare vom prezenta această structură de date. *BPB*-ul aflat în sectorul de boot definit în *MS-DOS* `2.x` permitea doar dimensiuni de cel mult 65535 sectoare (mai puțin de 32 MB în sectoare de câte 512 bytes). Această limitare se datora faptului că numărul total de sectoare era un câmp reprezentat pe 16 biți.

Începând cu *MS-DOS* `3.x` s-a modificat *BPB*-ul astfel încât să includă un câmp reprezentat pe 32 de biți, pentru a reține numătul total de sectoare.

Următoarea schimbare a *BPB*-ului a apărut la sistemul de operare *Windows '95 OSR2*, când a fost introdus *FAT32*. *FAT16* era limitat de mărimea maximă a *FAT*-ului și de mărimea maximă a unui *cluster* valid la nu mai mult de 2 GB dacă discul hard conținea sectoare de 512 bytes. *FAT32* elimină această limitare și astfel volumul (partiția) poate ocupa mai mult de 2 GB.

Primele câmpuri din *BPB*-ul *FAT32* (până la `BPB_TotSec32` inclusiv) sunt identice cu cele din *BPB*-ul *FAT12/FAT16*. Ele încep să difere de la offset-ul 36, în funcție de tipul de *FAT* folosit (vezi determinarea tipului de FAT).

Important este că *BPB*-ul aflat în sectorul de *boot* al unui *FAT* trebuie să conțină toate câmpurile noi ce apar față de *BPB*-ul *FAT12/FAT16* sau *FAT32*. Acest lucru asigură compatibilitatea unui program (*driver*) care trebuiie să înțeleagă și să suporte volumele corect, deoarece conține întotdeauna toate câmpurile definite.

În continuare vom prezenta sectorul de *boot*. Toate câmpurile al căror nume începe cu `BPB_` aparțin *BPB*-ului. Câmpurile al căror nume încep cu `BS_` fac parte din sectorul de *boot* și nu din `BPB`.

Primul câmp este `BS_jmpBoot`, apare la *offset*-ul 0 și are dimensiunea de trei bytes. El conține o instrucțiune de salt la codul de *boot*. Acest câmp poate avea două forme:

```
JmpBoot[0]=0x0EB,
jmpBoot[1]=0x??,
jmpBoot[2]=0x90
```

și:

```
JmpBoot[0]=0x0E9,
jmpBoot[1]=0x??,
jmpBoot[2]=0x??
```

Prin `0x??` se indică faptul că byte-ul respectiv poate conține orice valoare care poate fi reprezentată folosind opt biți.

Al doilea câmp este `BS_OEMName`, apare la *offset*-ul 3 și are dimensiunea de opt bytes. Acest câmp conține șirul `MSWIN4.1`. Sunt multe interpretări greșite ale acestui câmp; sistemele de operare de la *Microsoft* nu acordă atenție acestui câmp, în timp ce unele *driver*-e de FAT îl iau în considerare. Acesta este motivul pentru care șirul de caractere indicat este `MSWIN4.1`.

Al treilea câmp este `BPB_BytsPerSec`, apare la *offset*-ul 11 și are dimensiunea de doi bytes. Este folosit pentru a indica numărul de bytes care fac parte dintr-un sector. Valorile admise sunt 512, 1024, 2048 și 4096.

Al patrulea câmp este `BPB_SecPerClus`, apare la *offset*-ul 13 și are dimensiunea de un byte. Valoarea sa indică numărul de sectoare dintr-o unitate de alocare (*cluster*). Valoarea sa trebuie să fie o putere a lui 2, valorile permise fiind 1, 2, 4, 8, 16, 32, 64 și 128. Trebuie amintit faptul că un *cluster* are dimensiunea (în bytes) dată de produsul `BPB_BytsPerSec*BPB_SecPerClus`. Cea mai mare valoare permisă pentru mărimea unui *cluster* de 64 KB.

Al cincilea câmp este `BPB_RsvdSecCnt`, apare la *offset*-ul 14 și are dimensiunea de doi bytes. Valoarea sa indică numărul de sectoare din zona rezervată a volumului (partiției). Acest câmp nu trebuie să fie 0. Pentru *FAT12* și *FAT16* această valoare nu trebuie să fie diferită de 1, iar pentru *FAT32*, ea este de obicei 32.

Al șaselea câmp este `BPB_NumFATs`, apare la *offset*-ul 16 și are dimensiunea de un byte. El indică numărul de structuri *FAT* prezente pe volumul curent. Acest câmp trebuie să conțină întotdeauna valoarea 2 pentru orice volum de tip *FAT*. Cu toate acestea, orice valoare mai mare sau egală cu 1 este permisă. Foarte puține programe și sisteme de operare nu funcționează corect dacă sunt prezente structuri de *FAT* în număr diferit de 2.

Al șaptelea câmp este `BPB_RootEntCnt`, apare la *offset*-ul 17 și are dimensiunea de doi bytes. Pentru *FAT12* și *FAT16*, acest câmp conține numărul maxim de intrări permise în directorul rădăcină. Pentru *FAT32*, acest câmp trebuie să aibă valoarea 0. În cazul volumelor *FAT12* și *FAT16* acest câmp trebuie să aibă o valoare care, înmulțită cu 32, este un multiplu al valorii câmpului `BPB_BytsPerSec`. Pentru compatibilitate maximă, volumele *FAT16* ar trebui să folosească valoarea 512.

Al optulea câmp este `BPB_TotSec16`, apare la *offset*-ul 19 și are dimensiunea de doi bytes. Acest câmp conține numărul total de pe toate cele patru regiuni ale volumului. Dacă valoarea sa este 0, atunci valoarea câmpului `BPB_TotSec32` nu poate fi tot 0. Dacă valoarea acestui al doilea câmp este 0, atunci înseamnă că numărul total al sectoarelor este mai mic decât 65536, deci poate fi reprezentat folosind câmpul `BPB_TotSec16` care conține doar 16 biți.

Al nouălea câmp este `BPB_Media`, apare la *offset*-ul 21 și are dimensiunea de un byte. Pentru discurile fixe valoarea standard este `0xF8`. Pentru discurile mobile este folosită adesea valoarea `0xF0`. Valorile admise pentru acest câmp sunt `0xF0`, `0xF8`, `0xF9`, `0xFA`, `0xFB`, `0xFC`, `0xFD`, `0xFE` și `0xFF`. Singurul lucru important este acela că această valoare trebuie să apară în byte-ul cel mai nesemnificativ al primei intrări în *FAT*, `FAT[0]`.

Al zecelea câmp este `BPB_FATSz16`, apare la *offset*-ul 22 și are dimensiunea de doi bytes. Acest câmp reprezintă numărul de sectoare ocupate de un *FAT* pentru volumele *FAT12/FAT16*. Pe volumele *FAT32* acest câmp trebuie să conțină valoarea 0; câmpul `BPB_FATSz32` va conține mărimea în sectoare a unui *FAT*.

Al unsprezecelea câmp este `BPB_SecPerTrk`, apare la *offset*-ul 24 și are dimensiunea de doi bytes. Acest câmp conține numărul de sectoare pe pistă și este folosit de întreruperea `0x13`.

Al doisprezecelea câmp este `BPB_SecPerTrk`, apare la *offset*-ul 26 și are dimensiunea de doi bytes. Întreruperea `0x13` folosește acest câmp pentru a determina numărul de capete. Pentru o dischetă de 1.44 MB, valoarea sa este 2.

Al treisprezecelea câmp este `BPB_SecPerTrk`, apare la *offset*-ul 28 și are dimensiunea de patru bytes. Acest câmp conține numărul de sectoare ascunse care preced

partiția care conține acest volum *FAT*. Valoarea sa trebuie să fie 0 pentru medii care nu sunt partiționate.

Al paisprezecelea câmp este `BPB_TotSec32`, apare la *offset*-ul 32 și are dimensiunea de patru bytes. Acesta conține numărul de sectoare ale tuturor celor patru zone ale volumului. Valoarea sa poate fi 0; în acest caz câmpul `BPB_TotSec16` trebuie să aibă valoare diferită de 0. Pentru volumele *FAT32* valoarea acestui câmp este diferită de 0. Pentru volumele *FAT12/FAT16* acest câmp conține numărul de sectoare în cazul în care numărul total de sectoare depășește valoarea 65535; în acest caz valoarea câmpului `BPB_TotSec16` este 0.

Din acest punct (*offset*-ul 36) structura *BPB/boot* diferă pentru *FAT12/FAT16* și *FAT32*. Vom prezenta acum structura pentru volumele *FAT12* și *FAT16* începând cu *offset*-ul 36, al sectorului de *boot*.

Câmpul `BS_DrvNum` apare la *offset*-ul 36 și are dimensiunea de un byte. Întreruperea `0x13` folosește acest câmp pentru a determina o valoare caracteristică *drive*-ului. Pentru discuri hard valoarea este cuprinsă între `0x80` și `0x83` iar pentru dischete între `0x00` și `0x01`.

Câmpul `BS_Reserved1` apare la *offset*-ul 37 și are dimensiunea de un byte. Acest câmp este rezervat (folosit la formatare de *Windows NT*) și trebuie să aibă valoarea 0.

Câmpul `BS_BootSig` apare la *offset*-ul 38 și are dimensiunea de un byte. Valoarea sa este constanta `0x29` și indică prezența următoarelor trei câmpuri.

Câmpul `BS_VolID` apare la *offset*-ul 39 și are dimensiunea de patru bytes. Valoarea acestui câmp reprezintă numărul serial (*serial number*) al volumului. Este folosit, în general, pentru medii care nu sunt fixe, cum ar fi dischetele.

Câmpul `BS_VolLab` apare la offset-ul 43 și are dimensiunea de 11 bytes. El este folosit pentru a păstra eticheta de volum.

Câmpul `BS_FilSysType` apare la offset-ul 54 și are dimensiunea de opt bytes. Acest câmp conține unul din șirurile de caractere: `"FAT12   "`, `"FAT16   "` sau `"FAT     "` (fără ghilimele). Mulți cred că acest șir de caractere are legatură cu tipul de *FAT* folosit pe acest volum; acest lucru nu este însă adevărat.

Vom prezenta acum structura sistemului *FAT32* începând cu *offset*-ul 36.

Câmpul `BPB_FATSz32` apare la *offset*-ul 36 și are dimensiunea de patru bytes. Acest câmp este definit numai pentru *FAT32* și nu există pe volume *FAT12* și *FAT16*. El conține numărul de sectoare ocupate de un singur *FAT*. Valoarea câmpului `BPB_FATSz16` trebuie neapărat să fie 0.

Câmpul `BPB_ExtFlags` apare la *offset*-ul 40 și are dimensiunea de doi bytes. Acest câmp este și el definit numai pe volumele *FAT32*. Biții 0-3 indică numărul de structuri *FAT* active. Acest număr este valid dacă *mirroring*-ul este dezactivat. Biții 4-6 sunt rezervați. Dacă bitul are valoarea 0, înseamnă că *FAT*-ul este oglindit în toate FAT-urile definite. Biții 8-15 sunt rezervați.

Câmpul `BPB_FSVer` apare la *offset*-ul 42 și are dimensiunea de doi bytes. Acest câmp este, de asemenea, definit numai pe volumele *FAT32*. Byte-ul cel mai semnificativ reprezintă numărul major al reviziei și byte-ul cel mai puțin semnificativ reprezintă numărul minor al reviziei. Acest câmp reprezintă versiunea volumului *FAT32*. Versiunea curentă este 0:0. Dacă această valoare este diferită de 0, versiunile de *Windows* precedente nu vor monta acest volum.

Câmpul `BPB_RootClus` apare la *offset*-ul 44 și are dimensiunea de patru bytes. Acest câmp, definit numai pentru *FAT32*, reprezintă numărul *cluster*-ului la care începe directorul rădăcină.

Câmpul `BPB_FSInfo` apare la *offset*-ul 48 și are dimensiunea de doi bytes. Este și el definit numai pentru *FAT32* și reprezintă numărul sectorului unde se află structura *FSINFO*, în zona rezervată. De obicei acest câmp are valoarea 1.

Câmpul `BPB_BkBootSec` apare la *offset*-ul 50 și are dimensiunea de doi bytes. De asemenea, acest câmp este definit numai pe volumele *FAT32*; dacă acest câmp are valoarea diferită de 0, atunci reprezintă numărul sectorului din zona rezervată unde se află o copie a sectorului de *boot*. De obicei această valoare este 6; nu se recomandă folosirea unei alte valori.

Câmpul `BPB_Reserved` apare la *offset*-ul 52 și are dimensiunea de 12 bytes. Acesta este definit numai pe volumele *FAT32* și este rezervat pentru expansiunea viitoare a acestei structuri *FAT32*.

Câmpul `BS_DrvNum` apare la *offset*-ul 64 și are dimensiunea de un byte. Acest câmp are aceeași semnificație ca și pentru *FAT12* și *FAT16*. Singura diferență este faptul că pentru volumele FAT32 acest câmp apare la un alt *offset*.

Câmpul `BS_Reserved1` apare la *offset*-ul 65 și are dimensiunea de un byte. Acest câmp are și el aceeași semnificație ca și pentru *FAT12* și *FAT16*, singura diferență fiind, și în acest caz, faptul că pentru volumele *FAT32* apare la un alt *offset*.

Câmpul `BS_BootSig` apare la *offset*-ul 66 și are dimensiunea de un byte. Acest câmp are, de asemenea, aceeași semnificație ca și pentru *FAT12* și *FAT16*, dar apare la un alt *offset*.

Câmpul `BS_VolID` apare la *offset*-ul 67 și are dimensiunea de patru bytes. Acesta are aceeași semnificație ca și pentru *FAT12* și *FAT16*, singura diferență fiind *offset*-ul la care apare.

Câmpul `BS_VolID` apare la *offset*-ul 71 și are dimensiunea de 11 bytes. Din nou, semnificația este identică celei de la sistemele *FAT12* și *FAT16*, singura diferență fiind, și de această dată, *offset*-ul la care apare.

Câmpul `BS_FilSysType` apare la *offset*-ul 82 și are dimensiunea de opt bytes. Pentru sistemele *FAT32* valoarea sa este `"FAT32   "` (fără ghilimele), dar, la fel ca și în cazul sistemelor *FAT12* și *FAT16*, această valoare nu are nici o relevanță.

Mai trebuie adăugat un lucru în ceea ce privește sectorul 0 al unui volum *FAT*. Dacă am considera conținutul

sectorului ca un tablou de bytes (începând cu indicele 0), ar trebui ca `Sector[510]` să aibă valoarea `0x55` și `Sector[511]` să aibă valoarea cu `0xAA`.

Multe documentații referitoare la FAT afirmă, greșind bineînțeles, că semnătura `0xAA55` ocupă "*ultimii doi bytes ai sectorului de boot*". Această afirmație este corectă numai dacă valoarea câmpului `BPB_BytsPerSec` este 512. Dacă această valoare este mai mare decât 512, *offset*-ul acestei semnături nu se schimbă (cu toate că nu este greșit să se pună această semnătură și la sfârșitul sectorului).

Să presupunem că avem o partiție cu mărimea în sectoare `DskSz`. Este important să se verifice valoarea câmpului `BPB_TotSec16/32` (unul dintre cele două câmpuri este diferit de zero, cel diferit de zero se verifică); această valoare nu poate fi mai mare decât `DskSz`. Valoarea acestor câmpuri trebuie să fie mai mică sau egală cu `DskSz`, deoarece nu este nimic în neregulă cu volumul *FAT*.

De fapt, nu este nimic neobișnuit să avem `BPB_TotSec16/32` cu valoarea mai mică decât `DskSz`. Este în regulă dacă `BPB_TotSec16/32` este considerabil mai mic decât `DskSz`.

Aceasta înseamnă că se irosește spațiu pe disc și nu faptul că volumul *FAT* este alterat în vreun fel. Totuși, dacă `BPB_TotSec16/32` este mai mare decât `DskSz`, volumul este serios distrus sau malformat deoarece el se întinde peste limita discului sau partiția se suprapune peste datele care urmează pe disc, după sfârșitul ei. Dacă un volum pentru care `BPB_TotSec16/32` este prea mare pentru discul pe care apare este considet valid, pot apărea pierderi catastrofale de date.

Zona FAT

Următoarea structură importantă este *FAT*-ul însuși. Această structură de date definește o listă simplu înlănțuită a *cluster*-elor unui fișier. Trebuie remarcat faptul că, în acest punct, directorul din *FAT* sau un *container* de fișier nu sunt altceva decât simple fișiere. Singurul lucru care face diferența dintre un director și un fișier este acela că directorul are un atribut special care indică faptul că "*fișierul*" este director. Următorul lucru important despre directoare este că "*fișierul*" conține o serie de intrări în *FAT* și ocupă un spațiu de 32 de bytes. În toate celelalte privințe, directorul este similar unui fișier.

FAT-ul unui anumit volum este o hartă numerotată a *cluster*-elor, unde primul *cluster* de date este *cluster*-ul 2.

Primul sector al *cluster*-ului 2 (regiunea de date a discului) este stocat folosind câmpurile *BPB* pentru un volum. Pentru început trebuie determinată mărimea directorului rădăcină în număr de sectoare :

```
RootDirSectors=((BPB_RootEntCnt*32)+
 (BPB_BytsPerSec-1))/BPB_BytsPerSec;
```

Pe un volum *FAT32* câmpul `BPB_RootEntCnt` are întotdeauna valoarea `0`. Numărul `32` din formula precedentă reprezintă mărimea în bztes a unei intrări din *FAT* corespunzătoare unui director. A se observa, de asemenea, că rezultatul acestui calcul se rotunjește prin adaos.

Începutul zonei de date, primul sector al *cluster*-ului 2, se calculează în felul următor:

```
if(BPB_FATSz16!=0)
  FATSz=BPB_FATSz16;
else
  FATSz=BPB_FATSz32;
FirstDataSector=BPB_ResvdSecCnt+
  (BPB_NumFATs*FATSz)+RootDirSectors;
```

De observat că numărul acestui sector este relativ la primul sector al volumului care conține *BPB*-ul (sectorul care conține *BPB*-ul este sectorul cu numărul 0). Acesta nu trebuie să fie situat relativ la sectorul considerat ca fiind începutul partiției, deoarece, după partiționare, sectorul 0 al partiției nu este neapărat și sectorul 0 al *drive*-ului.

Fiind dat numărul *N* al unui *cluster* de date valid, numărul primului sector al *cluster*-ului (relativ la sectorul 0 al volumului *FAT*) se calculează în felul următor.

```
FirstSectorofCluster=((N-2)*
  BPB_SecPerClus)+FirstDataSector;
```

Deoarece `BPB_SecPerClus` este restricționat la puteri ale lui 2 (1, 2, 4, 8, 16, 32, ...), acest lucru înseamnă că înmulțirile și împărțirile cu `BPB_SecPerClus` pot fi realizate cu ajutorul deplasărilor pe biți care sunt mult mai rapide comparativ cu instrucțiunile `MUL` și `DIV`. Procesoarele cu arhitectură *Intel x86* execută cu aceeași viteză o instrucțiune `MUL` sau `DIV` cu o putere a lui 2 ca și o deplasare pe biți cu puterea lui 2.

Determinarea tipului de FAT

Apar foarte mari confuzii asupra modului exact de determinare a tipului de *FAT* care duc la erori de genul "*depășire cu* 1", "*depășire cu* 2", "*depășire cu* 10" sau "*depășire masivă*".

De fapt, modul de determinare este unul foarte simplu. Tipul de *FAT* (*FAT12*, *FAT16* sau *FAT32*) este determinat folosind doar numărul de *cluster*-e ale partiției și nimic altceva.

În continuare este recomandată citirea cu atenție a tuturor termenilor deoarece au o foarte mare importanță. De exemplu, termenul "*numărul de cluster-e*" este diferit de termenul "*numărul maxim de cluster-e valide*" deoarece primul *cluster* de date este 2, nu 0 sau 1.

Pentru început să vedem cum este determinată valoarea "*numărului de cluster-e*". Această valoare se determină cu ajutorul câmpurilor din *BPB*-ul partiției.

Pentru început se determină numărul de sectoare ocupate de directorul rădăcină, după cum s-a arătat anterior:

```
RootDirSectors=((BPB_RootEntCnt*32)+
  (BPB_BytsPerSec-1))/BPB_BytsPerSec;
```

Trebuie remarcat faptul că pe un volum *FAT32* câmpul `BPB_RootEntCnt` are întotdeauna valoarea `0`.

În continuare vom determina numărul de sectoare din regiunea de date a partiției:

```
if (BPB_FATSz16!=0)
  FATSz=BPB_FATSz16;
else
  FATSz=BPB_FATSz32;
if (BPB_TotSec16!=0)
  TotSec=BPB_TotSec16;
else
  TotSec=BPB_TotSec32;
DataSec=TotSec-(BPB_ResvdSecCnt+
  BPB_NumFATs*FATSz+RootDirSectors);
```

Acum vom determina numărul de *cluster*-e:

```
CountofClusters=DataSec/
  BPB_SecPerClus;
```

A se observa că acest calcul se rotunjește prin lipsă.

În acest moment putem determina tipul de *FAT*. Operațiile trebuie efectuate cu atenție deoarece este foarte posibil să apară o eroare de tipul "*depășire cu* 1".

În exemplul ce urmează vom descrie modul de determinare al tipului de *FAT*. Acolo unde se întâlnește semnul `"<"` nu înseamnă și `"<="`. Toate numerele de mai jos sunt corecte: primul număr (pentru *FAT12*) este 4085; al doilea număr (pentru *FAT16*) este 65525.

```
if (CountofClusters<4085){
  /* Volum FAT12 */
}
else
  if (CountofClusters<65525){
    /* Volum FAT16 */
  }
  else{
    /* Volum FAT32 */
  }
```

Aceasta este singura metodă de determinare a tipului de *FAT*. Nu există volume *FAT12* cu mai mult de 4084 *cluster*-e, nici volume (partiții) *FAT16* cu mai puțin de 4085 *cluster*-e sau mai mult de 65524 *cluster*-e precum nici volume (partiții) de *FAT32* cu mai puțin de 65525 *cluster*-e. Dacă veți încerca să realizați o partiție *FAT* care încalcă aceste reguli, sistemele de operare *Microsoft* nu vor funcționa corect deoarece vor considera că este vorba de un tip diferit de *FAT* față de cel considerat de dumneavoastră.

Se observă că valoarea `CountofClusters` este exact cea calculată mai sus. Cel mai mare număr al unui *cluster* de date valid pentru un volum este `CountOfClusters+1` și "*numărul total de cluster-e inclusiv cele două cluster-e rezervate*" este `CountofClusters+2`.

Mai avem un singur calcul important legat de *FAT*. Fiind dat numărul *N* al unui *cluster* valid, se va calcula unde este localizată în *FAT* intrarea pentru acest *cluster*. Acest lucru se calculează foarte ușor pentru *FAT16* și *FAT32*, calculul complicându-se pentru *FAT12*:

```
if (BPB_FATSz16!=0)
  FATSz=BPB_FATSz16;
else FATSz=BPB_FATSz32;
if (FATType==FAT16)
  FATOffset=N*2;
else
  if (FATType==FAT32)
    FATOffset=N*4;
ThisFATSecNum=BPB_ResvdSecCnt+
  (FATOffset/BPB_BytsPerSec);
ThisFATEntOffset=FATOffset%
  BPB_BytsPerSec;
```

În `ThisFATSecNum` este reținut numărul sectorului din primul *FAT* care conține intrarea pentru al *N*-lea *cluster* la *offset*-ul `ThisFATEntOffset`. Dacă se dorește calcularea numărului sectorului din al doilea *FAT*, se va aduna `FATSz` la `ThisFATSecNum`, iar pentru al treilea *FAT* se va aduna `2*FATSz` ș.a.m.d.

În acest moment se poate citi sectorul `ThisFATSecNum` (numărul acestui sector este relativ la sectorul 0 al volumului *FAT*). Considerăm că acest sector se citește folosind un tablou de bytes numit `SecBuff` și că `WORD` reprezintă un tip de date fără semn pe 16 biți, iar `DWORD` este un tip de date fără semn pe 32 de biți.

```
if (FATType==FAT16)
  FAT16ClusEntryVal=*((WORD*)
    &SecBuff[ThisFATEntOffset]);
else
  FAT32ClusEntryVal=(*((DWORD*)
    &SecBuff[ThisFATEntOffset]))
    &0x0FFFFFFF;
```

Modificarea conținutului unui *cluster* în *FAT16/32* se face în felul următor:

```
if (FATType==FAT16)
  *((WORD*)&SecBuff[ThisFATEntOffset])
    =FAT16ClusEntryVal;
else{
  FAT32ClusEntryVal=FAT32ClusEntryVal
    &0x0FFFFFFF;
  *((DWORD*)&SecBuff[ThisFATEntOffset])
  =(*((DWORD*)&SecBuff[ThisFATEntOffset]))
  &0xF0000000;
  *((DWORD*)&SecBuff[ThisFATEntOffset])
  =(*((DWORD*)&SecBuff[ThisFATEntOffset]))
   |FAT32ClusEntryVal;
}
```

Vom descrie modul de funcționare a codului de mai sus pentru *FAT32*. O intrare *FAT* pentru *FAT32* ocupă, de fapt, 28 de biți. Cei mai semnificativi patru biți ai unei intrări în *FAT* sunt rezervați. Cei patru biți trebuie modificați când se formatează volumul *FAT32* și atunci toate intrările în *FAT32* au cei mai semnificativi patru biți setați pe 0.

Aici trebuie dată o explicație suplimentară, pentru a evita apariția confuziilor. Pentru început trebuie reținut faptul că intrările pe 32 de biți în *FAT* sunt, de fapt, intrări pe 28 de biți. De exemplu, toate cele trei intrări pe 32 de biți `0x10000000`, `0xF0000000` și `0x00000000`, indică faptul că acel *cluster* este liber, deoarece se ignoră cei mai semnificativi patru biți când se citește valoarea intrării *cluster*-ului.

Dacă valoarea pe 32 de biți a valorii *cluster*-ului liber este `0x30000000` și doriți să îl marcați ca fiind un *cluster* "*bad*" atunci se va scrie valoarea `0x0FFFFFF7`. Intrarea pe 32 de biți va conține valoarea `0x3FFFFFF7`, deoarece trebuie să păstrați cei mai semnificativi patru biți, în momentul în care se scrie intrarea `0x0FFFFFF7` care reprezintă marcajul pentru un *cluster* "*bad*".

Deoarece `BPB_BytsPerSec` este întotdeauna divizibil cu 4, nu trebuie să ne îngrijoreze faptul că o intrare *FAT16/32* ar putea depăși un sector, deoarece acest lucru nu se poate întâmpla.

Codul sursă pentru *FAT12*, este mai complicat pentru că se folosesc 1,5 bytes (12 biți) pentru o intrare în *FAT*.

```
if (FATType==FAT12)
  FATOffset=N+(N/2);
ThisFATSecNum=BPB_ResvdSecCnt+
  (FATOffset/BPB_BytsPerSec);
ThisFATEntOffset=REM(FATOffset/
  BPB_BytsPerSec);
```

Acum trebuie verificat cazul în care se produce depășirea unui sector:

```
if (ThisFATEntOffset==(BPB_BytsPerSec
                                -1)){
// depasire de sector
}
```

În acest moment se accesează intrarea în *FAT* ca `WORD`, la fel ca pentru *FAT16*, dar dacă numărul *cluster*-ului este par atunci avem nevoie numai de cei mai puțin semnificativi 12 biți ai intrării de 16 biți pe care o prelucrăm; în cazul în care numărul *cluster*-ului este impar, atunci avem nevoie de cei mai semnificativi 12 biți din cei 16 cu care lucrăm.

```
FAT12ClusEntryVal=*((WORD*)
  &SecBuff[ThisFATEntOffset]);
if (N&0x0001)
  FAT12ClusEntryVal=
    FAT12ClusEntryVal>>4;
else
  FAT12ClusEntryVal=
    FAT12ClusEntryVal&0x0FFF;
```

Pentru a seta conținutul aceluiași *cluster* se folosește următorul cod:

```
if (N&0x0001){
  FAT12ClusEntryVal=
    FAT12ClusEntryVal<<4;
  *((WORD*)&SecBuff[ThisFATEntOffset])
    =(*((WORD*)&SecBuff[
    ThisFATEntOffset]))&0x000F;
}
else{
  FAT12ClusEntryVal
    =FAT12ClusEntryVal&0x0FFF;
  *((WORD*)&SecBuff[ThisFATEntOffset])
    =(*((WORD*)&SecBuff[
    ThisFATEntOffset]))&0xF000;
}
*((WORD*)&SecBuff[ThisFATEntOffset])
  =(*((WORD*)&SecBuff[
  ThisFATEntOffset]))
  |FAT12ClusEntryVal;
```

Operatorul `">>"` realizează deplasarea la dreapta pe biți și cei mai semnificativi patru biți primesc valoarea 0; operatorul `"<<"` realizează deplasarea la stânga pe biți și cei mai puțin semnificativi patru biți primesc valoarea 0.

Modul în care datele unui fișier sunt asociate cu fișierul este realizat astfel: în intrarea în director a fișierului/directorului este stocat numărul primului *cluster* de la care începe fișierul; primul *cluster* al fișierului este asociat cu datele conținute de primul *cluster* din fișier și localizarea acestor date pe partiție (volum) se determină folosind numărul *cluster*-ului.

Un fișier de dimensiune 0 (fișier care nu are alocate în nici un fel date) are numărul primului *cluster* 0, plasat în propria intrare din director. Locația acestui *cluster* în *FAT* conține fie marcajul de sfârșit de fișier (*EOC - End Of Clusterchain*), fie numărul următorului *cluster* din fișier. Valoarea *EOC* depinde de tipul *FAT*-ului (să presupunem că `FATContent` este conținutul intrării *cluster*-ului în *FAT* care va fi verificată dacă este sau nu *EOC*):

```
IsEOF=FALSE;
if (FATType==FAT12){
  if (FATContent>=0x0FF8)
    IsEOF=TRUE;
}
else
  if (FATType==FAT16){
    if (FATContent>=0xFFF8)
      IsEOF=TRUE;
  }
```

```
else
  if (FATType==FAT32){
    if (FATContent>=0x0FFFFFF8)
      IsEOF=TRUE;
  }
```

Numărul *cluster*-ului a cărui intrare în *FAT* conține *EOC* este alocat fișierului și este de asemenea ultimul *cluster* alocat fișierului.

Sistemele de operare *Microsoft* folosesc *driver*-e care utilizează pentru *EOC* valorile `0x0FFF` pentru *FAT12*, `0xFFFF` pentru *FAT16* și `0x0FFFFFFF` pentru *FAT32* când setează conținutul unui *cluster* din *FAT* cu valoarea *EOC*.

Există mai multe utilitare de disc pentru sistemele de operare *Microsoft* care setează sfârșitul de fișier cu altă valoare (între `0x0FF8` și `0x0FFF` pentru *FAT12*; `0xFFF8` și `0xFFFF` pentru *FAT16*; `0x0FFFFFF8` și `0x0FFFFFFF` pentru *FAT32*).

Mai există un marcaj special folosit pentru "*Bad Clusters*" (*cluster*-e avariate). Orice *cluster* care conține valoarea acestui marcaj nu va fi folosit pe post de *cluster* liber deoarece poate conduce la erori fatale ale discului. Acest marcaj are valorile: `0x0FF7` pentru *FAT12*, `0xFFF7` pentru *FAT16* și `0x0FFFFFF7` pentru *FAT32*.

Un alt lucru foarte important este că *bad*-urile sunt *cluster*-e "*pierdute*" - *cluster*-e care par să fie alocate deoarece nu conțin valoarea 0, dar care nu aparțin lanțului de alocare al nici unui fișier.

Observați că nu este posibil ca un *cluster* al cărui marcaj este "*bad*" să fie un *cluster* alocabil în *FAT12* și *FAT16* dar este posibil ca marcajul `0x0FFFFFF7` să fie numărul unui *cluster* alocabil pe volumele *FAT32*.

Pentru a evita posibila confuzie a utilitarelor de disc, nici un volum *FAT32* nu ar trebui niciodată configurat astfel încât valoarea `0x0FFFFFF7` să reprezinte numărul unui *cluster* alocabil.

Lista *cluster*-elor libere din *FAT* nu este nimic altceva decât lista tuturor *cluster*-elor care conțin valoarea zero în intrarea lor din *FAT*. Această valoare trebuie prelucrată după cum s-a explicat anterior; pentru orice altă valoare, intrarea în corespunzătoare *FAT* nu este liberă.

Acestă listă cu *cluster*-ele libere nu este stocată pe disc; trebuie să fie calculată când volumul este montat, prin scanarea *FAT*-ului pentru intrări care conțin valoarea 0. Pe volumele *FAT32*, sectorul `BPB_FSInfo` poate conține un număr valid de *cluster*-e libere ale volumului (mai multe informații despre sectorul `FSInfo` din *FAT32* veți găsi într-un articol viitor).

Acum să vedem ce reprezintă cele două *cluster*-e rezervate din *FAT* (*cluster*-ele 0 și 1).

Primul *cluster* rezervat, `FAT[0]` conține valoarea `BPB_Media` pe cei mai puțin semnificativi opt biți și restul biților au valoarea 1.

De exemplu, dacă `BPB_Media` are valoarea `0xF8`, pentru *FAT12* avem `FAT[0]=0x0FF8`, pentru *FAT16* avem `FAT[0]=0xFFF8` și pentru *FAT32* avem `FAT[0]=0x0FFFFFF8`.

Al doilea *cluster* rezervat, `FAT[1]`, este setat la formatare cu valoarea *EOC*.

În volumele *FAT12*, acest *cluster* nu este folosit și conține întotdeauna valoarea *EOC*.

Pentru *FAT16* și *FAT32*, *driver*-ul sistemului de fișiere poate folosi cei mai semnificativi 2 biți din intrarea `FAT[1]` ca *flag*-uri de volum (partiție) care indică faptul că volumul (partiția) nu a fost "demontat" înainte de oprirea sistemului (ceilalți biți au întotdeauna valoarea 1).

Pentru *FAT16* avem `ClnShutBitMask=0x8000` și `HrdErrBitMask=0x4000`.

Pentru *FAT32* avem `ClnShutBitMask=0x08000000` și `HrdErrBitMask=0x04000000`.

Dacă valoarea `ClnShutBitMask&FAT[1]` este 1, înseamnă că volumul este "curat".

Dacă acest bit are valoarea este 0, înseamnă că *driver*-ul sistemului de fișiere nu a "demontat" volumul corect, după ultima utlizare.

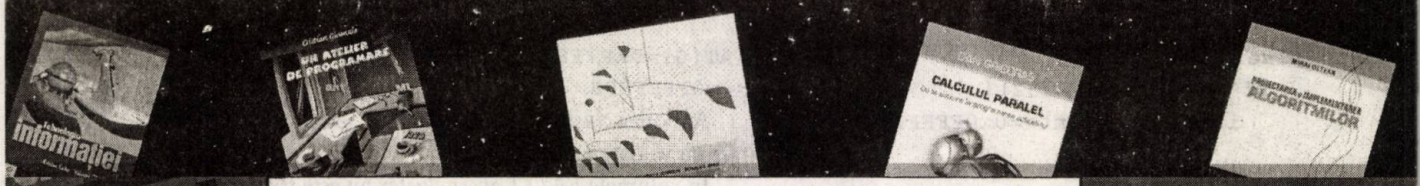
Dacă valoarea `HrdErrBitMask&FAT[1]` este 1 înseamnă că nu s-au semnalat erori de citire/scriere în decursul sesiunii de lucru anterioare.

Dacă acest bit are valoarea 0, înseamnă că *driver*-ul sistemului de fișiere a întâlnit erori de citire/scriere pe volum în timpul ultimei utilizări, ceea ce indică posibilitatea existenței unor sectoare "*bad*" pe volumul respectiv.

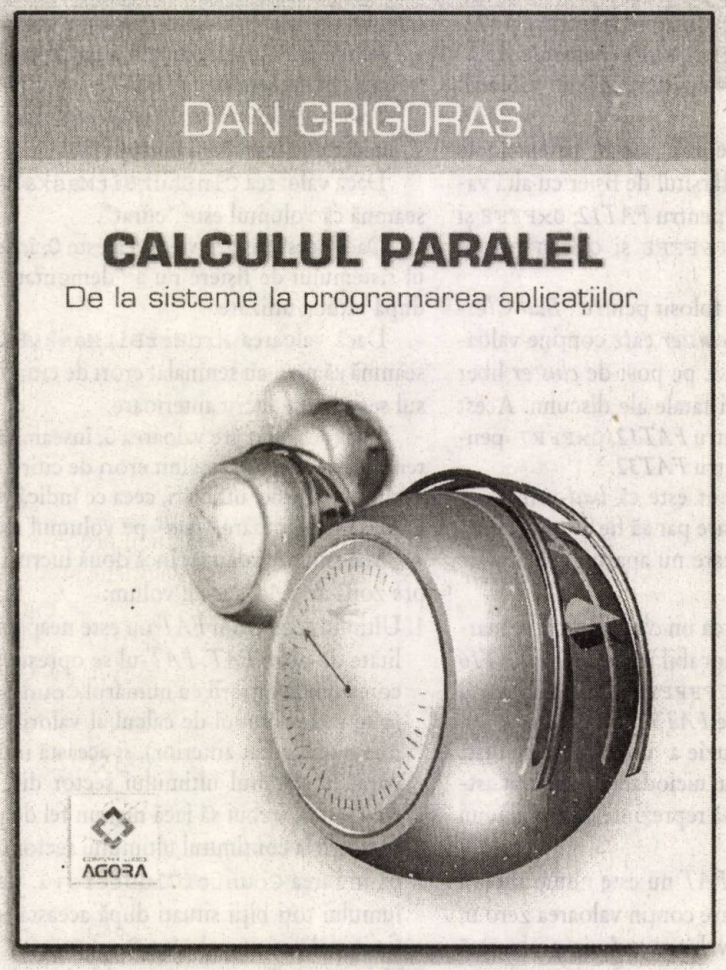
Mai sunt de adăugat încă două lucruri importante despre zona de *FAT* a unui volum:

1. Ultimul sector din *FAT* nu este neapărat folosit în totalitate de către *FAT*. *FAT*-ul se oprește la *cluster*-ul care corespunde intrării cu numărul `CountofClusters+1` (a se vedea modul de calcul al valorii `CountofClusters` prezentat anterior), și această intrare nu este neapărat la sfârșitul ultimului sector din *FAT*. Codul de *FAT* nu ar trebui să facă nici un fel de presupuneri care se referă la conținutul ultimului sector de *FAT*, aflat după intrarea `CountofClusters+1`. La formatarea volumului toți biții situați după această intrare trebuie să fie inițializați cu valoarea 0.
2. Valoarea `BPB_FATSz16` (respectiv `BPB_FATSz32` pentru *FAT32*) poate fi mai mare decât este necesar. Cu alte cuvinte, la sfârșitul *FAT*-urilor pot exista sectoare neutilizate. Din acest motiv ultimul sector folosit al *FAT*-ului este întotdeauna calculat folosindu-se valoarea `CountofCluster+1` și niciodată folosind valoarea `BPB_FATSz16/32`. Codul de *FAT* nu ar trebui să facă nici un fel de presupuneri referitoare la conținutul acestor sectoare suplimentare din *FAT*. La formatarea volumului, aceste sectoare suplimentare ar trebui inițializate cu valoarea 0.

Ciprian Ileană și Claudiu Soroiu sunt studenți în anul I la Universitatea Babeș-Bolyai din Cluj. Pot fi contactați la adresele **ciprian_il@yahoo.com**, *respectiv* **csoroiu@yahoo.com**.



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

AMD revine

Mihai Scorțaru

În ultima perioadă de timp AMD și-a întărit poziția pe piața procesoarelor destinate sistemelor desktop prin lansarea de noi procesoare destinate atât calculatoarelor ieftine, cât și celor performante. Intel s-a concentrat mai mult asupra pieții procesoarelor destinate calculatoarelor mobile, atingând bariera de 1 GHz și pentru aceste produse. Lansarea de noi produse a determinat și scăderea prețurilor celorlalte procesoare disponibile.

Noi produse

La târgul internațional *CeBit 2001* de la *Hanovra*, *AMD* a anunțat disponibilitatea pe piață a două noi procesoare din familia *Athlon*. Cele două procesoare au frecvențele de 1,3 și 1,33 GHz, principale diferență dintre ele fiind frecvența magistralei (200 MHz pentru primul și 266 MHz pentru al doilea).

Tot la *CeBit 2001*, *AMD* a demonstrat funcționarea unui procesor care avea frecvența de 1,5 GHz și făcea parte din următoarea generație a familiei *Athlon*. Numele de cod al acestuia este *Palomino*.

Pe data de 2 aprilie, *AMD* a făcut următorul pas logic și în ceea ce privește familia de procesoare *Duron*, lansând pe piață produsul *Duron 900*.

Chiar dacă lansarea pe piață a procesorului *Pentium 4* cu frecvența de 1,7 GHz este iminentă, *Intel* a preferat să mai amâne data acesteia până spre sfârșitul lunii aprilie. În schimb, compania a anunțat disponibilitatea produselor *Mobile Pentium III* cu frecvența de 1 GHz și *Mobile Celeron* cu frecvența de 750 MHz. De asemenea, pe data de 9 aprilie a fost lansat și un procesor *Celeron* cu frecvența de 850 MHz.

Atingerea barierei de 1 GHz de către procesoarele pentru calculatoarele portabile a avut loc la peste un an de la atingerea aceleiași bariere de către procesoarele pentru *desktop*-uri.

În acest moment *Intel* domină această piața deoarece procesoarele de la *AMD* nu au atins decât frecvența de 700 MHz.

Intel

Prețurile procesoarelor *Intel* s-au redus semnificativ, mai ales în ceea ce privește familia *Pentium 4*. Totuși, pentru procesoare la aceeași frecvență, prețurile practicate de *AMD* sunt mai mici. Până la sfârțitul lunii mai prețurile procesoarelor *Intel* se vor reduce în trei etape. La începutul lunii iunie, procesoarele *Intel* vor costa cu aproximativ 60% mai puțin.

Celeron

Apariția pe piață a unui nou procesor *Celeron* a determinat scăderea prețurilor celorlalte produse din această familie. La fel ca și *Celeron 800*, acest nou procesor are frecvența magistralei de 100 MHz.

La începutul lunii aprilie prețurile procesoarelor *Celeron* erau următoarele:

Celeron 533	*55$*	*-2$*	*(-4%)*
Celeron 566	*46$*	*-8$*	*(-15%)*
Celeron 600	*53$*	*-8$*	*(-13%)*
Celeron 633	*61$*	*+2$*	*(+3%)*
Celeron 667	*49$*	*-20$*	*(-29%)*
Celeron 700	*68$*	*-7$*	*(-9%)*
Celeron 733	*74$*	*-9$*	*(-11%)*
Celeron 766	*80$*	*-30$*	*(-27%)*
Celeron 800	*100$*	*-38$*	*(-28%)*
Celeron 850	*138$*	*n/a*	*n/a*

Toate prețurile procesoarelor *Celeron* au scăzut destul de mult, cu excepția celui cu frecvența de 633 MHz. Luna trecută prețul era mai mic deatorită stocurilor foarte mari. Se pare că, datorită cantității mari de procesoare de acest tip vândute, comercianții au mărit puțin prețul.

De data aceasta surpriza o consituie procesorul *Celeron 667* care costă doar 49$. O explicație a acestei reduceri neașteptate a prețului ar putea fi faptul că anumiți comercianți dețin multe astfel de procesoare și doresc să își lichideze stocurile. Este foarte probabil ca în scurt timp prețul acestui procesor să crească până la aproximativ 60$.

Pentium III

Prețurile procesoarelor *Pentium III* cu magistrala de 133 MHz sunt următoarele:

Pentium III 533	*119$*	*+19$*	*(+19%)*
Pentium III 600	*96$*	*-8$*	*(-8%)*
Pentium III 667	*98$*	*-15$*	*(-13%)*
Pentium III 733	*117$*	*-28$*	*(-19%)*
Pentium III 800	*145$*	*-25$*	*(-15%)*
Pentium III 866	*162$*	*-17$*	*(-9%)*
Pentium III 933	*204$*	*-11$*	*(-5%)*
Pentium III 1000	*224$*	*-30$*	*(-12%)*

Procesorul *Pentium III* la 1 GHz este acum mai scump decât un *Athlon* la 1,33 GHz. Majoritatea prețurilor au scăzut, cu excepția celor ale procesoarelor cu frecvențe de 533 MHz. Procesorul *Pentium III 533* a devenit o raritate, ceea ce explică și creșterea cu 19% a prețului său. Se pare că în acest moment frecvența de 667 MHz reprezintă o alegere foarte bună.

În cursul lunii aprilie va reveni pe piață și procesorul *Pentium III* cu frecvența de 1,13 GHz. După eșecul din vara anului 2000, când a fost lansat pentru prima dată acest procesor, Intel speră că a reușit să rezolve problemele apărute. Este foarte posibil ca acesta să fie ultimul procesor *Pentium III* fabricat pe baza tehnologiei de 0.18 microni. Viitoarele procesoare din această familie vor folosi tehnologia de 0.13 microni și vor avea frecvențe de peste 1,26 GHz.

Pentium 4

La aceste procesoare se observă cele mai mari reduceri. Prețurile sunt următoarele:

Pentium 4 1300	*277$*	*-65$*	*(-19%)*
Pentium 4 1400	*299$*	*-51$*	*(-15%)*
Pentium 4 1500	*475$*	*-94$*	*(-17%)*

Numărul procesoarelor *Pentium 4* disponibile pe piață a crecut foarte mult, ceea ce a dus la o reducere semnificativă a prețului acestora.

Procesorul *Pentium 4* cu frecvența de 1,7 GHz va fi lansat pe piață spre sfârșitul lunii aprilie.

AMD

Lansarea noilor procesoare *AMD* a determinat o reducere a prețurilor celorlalte. Noile procesoare Athlon cu frecvențele de 1,3 și 1,33 GHz au prețurile pe care le aveau înainte cele cu frecvența de 1,2 GHz.

Duron

Procesorul *Duron 900* a intrat pe piață la un preț de sub 100$. Din acest motiv, prețurile celorlalte procesoare din această familie au scăzut semnificativ. În Statele Unite procesoarele *AMD Duron* pot fi achiziționate la următoarele prețuri:

Duron 600	*42$*	*-3$*	*(-7%)*
Duron 650	*41$*	*-5$*	*(-11%)*
Duron 700	*41$*	*-7$*	*(-15%)*
Duron 750	*41$*	*-15$*	*(-27%)*
Duron 800	*53$*	*-14$*	*(-21%)*
Duron 850	*75$*	*-12$*	*(-14%)*
Duron 900	*92$*	*n/a*	*n/a*

În scurt timp va apărea și succesorul actualului procesor *Duron*. Acesta are numele de cod *Morgan* și, probabil, va avea frecvențe de cel puțin 950 MHz.

Athlon/200

Prețurile procesoarelor *Athlon Thunderbird* cu frecvența magistralei de 200 MHz au scăzut și ele datorită lansării procesorului *Athlon* cu frecvența de 1,3 GHz. Prețurile sunt următoarele:

Athlon 700	*64$*	*-13$*	*(-17%)*
Athlon 750	*76$*	*+4$*	*(+6%)*
Athlon 800	*91$*	*-8$*	*(-8%)*
Athlon 850	*90$*	*-11$*	*(-11%)*
Athlon 900	*107$*	*-15$*	*(-12%)*
Athlon 950	*136$*	*-11$*	*(-7%)*
Athlon 1000	*137$*	*-36$*	*(-21%)*
Athlon 1100	*149$*	*-50$*	*(-25%)*
Athlon 1200	*187$*	*-59$*	*(-24%)*
Athlon 1300	*216$*	*n/a*	*n/a*

În curând vor fi lansate pe piață procesoare Athlon cu frecvențe de 1,4 și 1,5 GHz, ceea ce va duce la noi scăderi de prețuri.

Athlon/266

Prețurile procesoarelor *Athlon Thunderbird* cu frecvența magistralei de 266 MHz au scăzut și ele datorită lansării procesorului *Athlon* cu frecvența de 1,33 GHz. Prețurile minime în Statele Unite sunt următoarele:

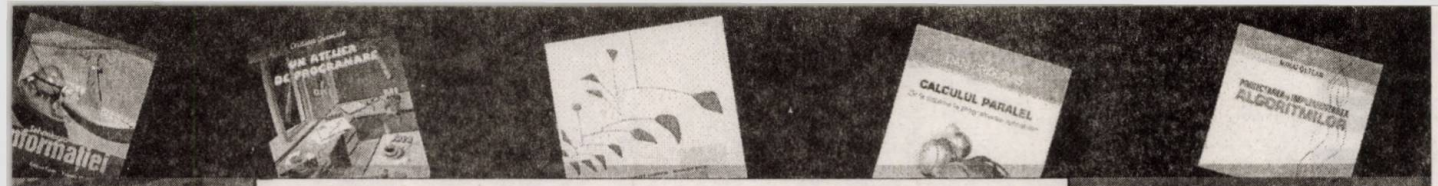
Athlon 1000	*148$*	*-62$*	*(-30%)*
Athlon 1133	*175$*	*-69$*	*(-28%)*
Athlon 1200	*192$*	*-57$*	*(-23%)*
Athlon 1333	*217$*	*n/a*	*n/a*

Viitoarele procesoare *Athlon* cu magistrala de 266 MHz poartă numele de cod *Palomino* și vor fi lansate la frecvențe de 1,53 GHz, 1,6 GHz și 1,67 GHz.

Concluzie

AMD continuă să ofere produse mai ieftine și mai performante decât cele de la *Intel*. Prețurile vor continua să scadă într-un ritm mai alert datorită faptului că atât *Intel*, cât și *AMD*, pregătesc lansarea de noi produse.

Mihai Scorțaru este redactor-șef adjunct al GInfo. Poate fi contactat prin e-mail la adresa **skortzy@xnet.ro**.



COMPUTER LIBRIS AGORA

Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Inteligența ARTIFICIALĂ

Mihai Scorțaru

În cadrul acestui articol vom încerca să demonstrăm, dacă mai este nevoie, că inteligența artificială reprezintă un domeniu care merită studiat. De asemenea, vom încerca să aflăm cu ce anume se ocupă acest domeniu și modul în care el a apărut.

Domeniul de studiu al *inteligenței artificiale* (*IA*) îl reprezintă înțelegerea entităților inteligente. Datorită acestui fapt, unul dintre motivele pentru care se fac cercetări în acest domeniu este acela de a învăța mai multe despre noi înșine. Spre deosebire de filozofie și psihologie, care studiază de asemenea inteligența, IA se ocupă în principal cu crearea și înțelegerea entităților inteligente.

Inteligența artificială a dus la obținerea unor produse foarte interesante, încă din primele momente ale existenței sale. Cu toate că nimeni nu poate prezice exact viitorul IA, este evident că apariția unor calculatoare cu un nivel de inteligență comparabil cu cel uman ar avea un impact uriaș asupra societății noastre.

Inteligența artificială încearcă să găsească răspunsul la una dintre întrebările fundamentale ale omenirii: *Cum este posibil ca un creier mic și lent, biologic sau electronic, să poată înțelege, prezice și manipula o lume mult mai vastă și mai complicată?* O altă întrebare al cărei răspuns este greu de găsit a fi: *Cum putem crea ceva care să aibă aceste proprietăți?* Răspunsul la aceste întrebări se lasă de mult timp așteptat dar, spre deosebire de călătoria cu viteze superluminice sau obținerea antigravitației, există motive suficiente să credem că scopul inteligenței artificiale poate fi atins. Pentru a găsi o dovadă concludentă în acest sens trebuie doar să ne privim în oglindă.

Inteligența artificială este una dintre cele mai noi discipline. Oficial, anul apariției sale este 1956, an care reprezintă momentul în care a fost ales numele său. Împreună cu genetica modernă, IA reprezintă domeniul de care cei mai mulți oameni de știință ar dori să se ocupe.

Studiul inteligenței este însă una dintre cele mai vechi activități ale omenirii. De peste 2000 de ani filozofii încearcă să înțeleagă modul în care funcționează simțurile, capacitatea de a învăța, de a memora etc.

În momentul de față IA conține numeroase subdomenii de studiu, de la cele generale care se ocupă cu gândirea logică până la unele particulare cum ar fi jocul de șah, demonstrarea unor teoreme matematice, scrierea de poezii sau diagnosticarea unor boli.

Ce este inteligența artificială?

Există numeroase definiții ale inteligenței artificiale. Formele acestora diferă în funcție de punctul de vedere din care este privit domeniul. Unele se axează pe procesele de gândire și pe raționamente, în timp ce altele se axează pe comportamente. De asemenea, unele folosesc pentru comparație inteligența umană, în timp ce altele se bazează pe un concept ideal de inteligență. Folosind aceste două criterii, obținem patru clase de definiții.

Definițiile din prima clasă se pot rezuma la ***sisteme care gândesc ca oamenii.*** Două dintre aceste definiții sunt:

- încercarea de a construi calculatoare care gândesc în întregul sens al cuvântului;
- automatizarea activităților pe care le asociem cu gândirea umană și anume luarea deciziilor, rezolvarea problemelor, învățarea etc.

Definițiile din a doua clasă se pot rezuma la ***sisteme care gândesc rațional.*** Două dintre aceste definiții sunt:

- studiul facultăților mintale folosind modele computaționale;
- studiul operațiilor computaționale care permit percepția, raționea și acțiunea.

Definițiile din a treia clasă se pot rezuma la ***sisteme care acționează ca oamenii.*** Două dintre aceste definiții sunt:

- arta creării mașinilor care efectuează operații pentru care este nevoie de inteligență atunci când sunt efectuate de oameni;
- construirea unor calculatoare care să efectueze operații pe care oamenii le realizează acum cu mai mult succes.

Definițiile din a patra clasă se pot rezuma la ***sisteme care acționează rațional.*** Două dintre aceste definiții sunt:

- un domeniu care încearcă să explice și să emuleze comportamentul inteligent în termeni de procese computaționale;
- ramura informaticii care se ocupă cu automatizarea comportamentului inteligent.

Toate cele patru abordări au fost urmate de-a lungul ultimelor decenii. Conform așteptărilor, există disensiuni între cei care folosesc abordări care se bazează pe inteligența umană și cei care folosesc abordări care se bazează pe rațiune.

Există numeroase alte definiții; câteva dintre ele sunt:
- o colecție de algoritmi accesibili (polinomiali) care să reprezinte aproximări ale unor probleme inaccesibile (pentru care nu există soluții polinomiale);
- construirea unui sistem fizic care să fie capabil să treacă testul *Turing*;
- ramura informaticii care studiază modul în care pot fi create mașini care acționează inteligent;
- un domeniu care folosește tehnici computaționale pentru a efectua sarcini care, aparent, necesită inteligență atunci când sunt realizate de oameni;
- o investigație foarte generală a naturii inteligenței și a principiilor și mecanismelor necesare pentru a o înțelege și a o duplica;
- programarea calculatoarelor astfel încât să efectueze operații care par a fi inteligente.

Comportamentul uman

Testul *Turing* a fost propus de *Alan Turing* în anul 1950 și a fost creat pentru a obține o definiție satisfăcătoare a inteligenței. Autorul a definit comportamentul inteligent ca fiind ***abilitatea de a obține performanțe care se ridică la nivelul celor umane în toate domeniile cognitive, astfel încât un interogator să poată fi păcălit.*** Cu alte cuvinte, un calculator ar deveni inteligent în momentul în care ar putea răspunde la anumite întrebări astfel încât un interogator să nu-și poată da seama dacă vorbește cu un om sau cu o mașină.

Pentru a putea trece un astfel de test, un calculator ar trebui să aibă următoarele caracteristici:
- *procesarea limbajului natural* pentru a permite comunicare într-o numită limbă;
- *reprezentarea cunoașterii* pentru a putea păstra informații primite înaintea sau în timpul interogării;
- *raționare automată* pentru a folosi informațiile stocate pentru a răspunde la întrebări și pentru a trage noi concluzii;
- *învățare* pentru a se adapta la noi circumstanțe și pentru a detecta și extrapola anumite șabloane.

Testul *Turing* nu implică un contact vizual între interogator și persoana sau calculatorul pe care îl interoghează, deoarece simularea fizică a unei persoane nu este neapărat necesară pentru a obține inteligența. Totuși, interogatorul poate testa abilitățile perceptuale ale subiectului prin intermediul unui semnal video sau a unor obiecte pe care i le poate trimite prin intermediul unei trape. Datorită acestui fapt, calculatorul ar trebui să aibă și următoarele caracteristici:
- percepție vizuală pentru a putea recunoaște obiectele;
- elemente de robotică pentru a le putea manipula.

În domeniul inteligenței artificiale nu s-au depus eforturi considerabile în încercarea de a trece testul *Turing*. Problema comportamentului uman va apărea doar în momentul în care programele de inteligență artificială vor trebui să interacționeze cu oamenii.

Gândirea umană

Pentru a putea spune că un calculator gândește ca un om, trebuie mai întâi să știm cum gândesc oamenii. Pentru aceasta va trebui să cercetăm cum funcționează mintea umană. Există două metode pentru a realiza acest lucru: ***introspecția*** (studierea propriilor gânduri pe măsură ce apar) și ***experimentele psihologice.***

Doar în momentul în care vom avea o teorie suficient de precisă asupra minții, vom putea încerca să o exprimăm cu ajutorul unui program pentru calculator.

Gândirea rațională

Filozoful grec *Aristotel* a fost primul care a încercat să definească gândirea corectă (rațională). Faimoasele sale silogisme oferă șabloane cu ajutorul cărora se ajunge la concluzii corecte dacă ipotezele sunt corecte. Idei asemănătoare au apărut la sfârșitul secolului al XIX-lea, o dată cu apariția logicii formale. Această logică a oferit notații precise pentru propoziții referitoare la o mulțime de elemente ale lumii și la relațiile dintre ele. În 1965 existau deja programe care, dacă dispuneau de suficient timp și suficientă memorie, preluau o descriere a unei probleme folosind notații logice și determinau soluția acesteia, dacă aceasta exista. În cazul în care nu exista nici o soluție, programele nu se opreau niciodată din căutare. Adepții așa numitei ***tradiții logiciste*** a inteligenței artificiale speră să obțină sisteme inteligente folosind astfel de programe.

Există cel puțin două dificultăți majore în fața acestei abordări. Prima dintre ele constă în faptul că este foarte dificil să transpunem cunoștințele în termeni necesari notației logice, mai ales când cunoștințele nu sunt 100% corecte. În al doilea rând, există o mare diferență între a rezolva o problemă în principiu și a o rezolva în practică. Chiar și problemele care folosesc un număr relativ mic de fapte (de ordinul zecilor) vor epuiza foarte rapid resursele computaționale ale oricărui calculator dacă nu există o "ghidare" asupra pașilor care trebuie urmați în raționament.

Comportamentul rațional

A acționa rațional înseamnă a ajunge la un anumit scop, date fiind anumite convingeri. Un ***agent rațional*** este o entitate care percepe și acționează. Această abordare consideră IA ca fiind știința care se ocupă cu studiul și construirea agenților rationali. Abordarea anterioară se baza, în primul rând, pe deducții (inferențe) corecte. Realizarea unor deducții corecte este, uneori, una dintre caracteristi-

cile agenților inteligenți deoarece o modalitate de a acționa rațional constă în a raționa logic spre concluzia că o anumită acțiune va duce la atingerea scopului și va acționa potrivit acestei concluzii. Totuși, deducerile corecte nu reprezintă în întregime rațiunea deoarece există deseori situații în care nu există nici o acțiune corectă, dar totuși trebuie să se acționeze într-un anume fel. De asemenea, există acțiuni raționale despre care nu se poate spune că implică deducții. Un exemplu ar fi retragerea mâinii în momentul în care atingem o suprafață încinsă. Acesta este un gest reflex care este mult mai eficient decât o acțiune mai lentă care ar implica o deliberare atentă asupra situației.

Toate capacitățile cognitive necesare pentru testul *Turing* sunt cerute pentru a putea acționa rațional. Studiul inteligenței artificiale folosind această abordare are două avantaje. În primul rând, comportamentul rațional este mai general decât gândirea rațională deoarece gândirea rațională este doar un mecanism (care uneori nu este nici necesar) folosit pentru a acționa rațional. În al doilea rând, este mai potrivită pentru dezvoltarea științifică decât abordările bazate pe comportamentul uman sau gândirea umană, deoarece standardul rațiunii este clar definit și complet general. Comportamentul uman este bine-adaptat pentru un mediu specific; el este, parțial, produsul unui complicat proces evoluționist care este departe de a ajunge la perfecțiune.

Bazele inteligenței artificiale

Cu toate că IA este un domeniu foarte nou, el a moștenit multe idei, puncte de vedere și tehnici caracteristice altor discipline. Datorită celor 2000 de ani în care a fost studiată filozofia, au apărut numeroase teorii referitoare la rațiune și învățare; de asemenea, a apărut punctul de vedere potrivit căruia mintea este efectul funcționării unui sistem fizic. Ultimii 1200 de ani de matematică au dus la apariția teoriei logicii formale, a teoriei probabilităților și a calculelor moderne. Din psihologie au fost preluate unelte pentru investigarea minții umane și un limbaj științific cu ajutorul cărora să poată fi exprimate teoriile obținute. Din lingvistică avem teoriile de structură și semnificație a limbajului. În sfârșit, informatica pune la dispoziție uneltele necesare pentru ca IA să devină realitate.

Filozofia

Se spune că întreaga tradiție filozofică europeană constă într-o serie de note de subsol care conțin referințe la operele lui *Platon*. Acesta s-a născut în anul 428 î.Hr., iar lucrările sale s-au concentrat pe diferite domenii cum ar fi filozofie, astronomie, fizică, matematică și politică. Împreună cu *Socrates*, profesorul său, *Aristotel*, elevul său, a pus bazele gândirii și culturii vestice. Se spune că începuturile inteligenței artificiale au fost în jurul anului 450 î.Hr. În unul dintre dialogurile sale, *Platon* spunea că, la un moment dat, *Socrates* i-a spus lui *Euthyphro*: "*Vreau să știu care este caracteristica bunătății care face toate acțiunile să fie bune... în ce ar trebui să o transform ca să o folosesc ca un standard pentru a judeca acțiunile tale și ale altora*". Cu alte cuvinte se cerea un algoritm cu ajutorul căruia să se poată deosebi acțiunile corecte de cele incorecte. *Aristotel* a continuat cu formularea unor legi mai exacte care guvernează partea rațională a minții. El a construit un sistem de silogisme pentru raționamente corecte care, în principiu, permite obținerea unor concluzii corecte pe baza unor premize inițiale. Aristotel nu credea că toate componentele minții erau guvernate de procese logice; el a introdus și noțiunea de rațiune intuitivă.

Următorul pas a fost considerarea minții ca fiind un proces fizic. *René Descartes* (1596 - 1650) a fost cel care a pus în discuție distincția dintre minte și materie. Una dintre problemele cu care se confrunta conceptul potrivit căruia mintea este guvernată în întregime de legi fizice este aceea că, în acest fel, libera voință nu ar mai avea sens. Astfel oamenii ar avea la fel de multă voință ca și *Pământul* care "decide" să se învârtă în jurul *Soarelui*. *Descartes* a introdus conceptul ***dualismului*** potrivit căruia o parte a minții se află în afara naturii și nu se supune legilor fizice. El a afirmat că această parte poate fi considerată ca fiind ***sufletul*** sau ***spiritul***. Tot el a afirmat că animalele nu posedă această calitate duală, motiv pentru care ele pot fi considerate ca fiind mașini.

O alternativă a dualismului a fost ***materialismul*** care susținea că întreaga lume, inclusiv creierul și mintea, funcționează pe baza unor legi fizice. Cel care a încercat pentru prima dată să construiască un dispozitiv mecanic care să aibă capacități mentale a fost *Wilhelm Leibniz* (1646 - 1716). Din nefericire el nu a obținut rezultate semnificative.

Se poate adopta și o poziție intermediară potrivit căreia se acceptă faptul că mintea are o bază fizică dar se neagă faptul că acestea pot fi explicate folosind reduceri la procese fizice obișnuite. Potrivit acestei concepții procesele mentale și conștiința fac parte din lumea fizică, dar sunt mai presus decât capacitatea de înțelegere a omului. Această poziție a fost adoptată de unii filozofi care se împotriveau dezvoltării inteligenței artificiale.

Eliminând unele obiecții, se poate spune că filozofia consideră mintea ca fiind un dispozitiv fizic care funcționează, în principal, cu ajutorul rațiunii și folosind cunoștințele acumulate. Următoarea problemă care apare este stabilirea sursei cunoașterii.

Mișcarea ***empiristă*** se caracterizează prin afirmația lui *John Locke* (1632 - 1701): "*Nimic nu ajunge în sfera cunoașterii înainte să treacă prin simțuri*". *David Home* a propus actualul principiu al inducției potrivit căruia regulile generale pot fi deduse folosind diferite asociații între elementele lor. *Bertrand Russel* a dezvoltat această teorie și a creat ***pozitivismul logic***. Această doctrină susține că toate cunoștințele pot fi caracterizate prin teorii logice înlănțuite până se ajunge la ***propoziții de observație*** care corespund ***intrărilor senzoriale*** (organe de simț). ***Teoria confirmării***, introdusă de *Rudolf Carnap* și *Carl Hempel*, a încercat să explice cum poate fi obținută cunoașterea pe baza experienței.

Ultimul element al imaginii filozofice a minții îl reprezintă legătura dintre cunoaștere și acțiune. Două dintre întrebările vitale din cadrul IA sunt *cum se realizează această legătură* și *cum pot fi justificate diferite acțiuni particulare*. Doar înțelegând justificarea acțiunilor am putea construi un agent ale cărui acțiuni să fie justificabile sau raționale.

Matematica

Chiar dacă filozofii au schițat cele mai importante idei ale inteligenței artificiale, pentru a crea o știință exactă este nevoie de claritate matematică în cel puțin trei domenii: computațional, logic și probabilistic. Ideea de a exprima un calcul computațional sub forma unui algoritm i-a venit, pentru prima dată, lui *al-Khowarazmi*, un matematician arab care a trăit în secolul al IX-lea. *George Boole* (1815 - 1864) a fost cel care a introdus logica matematică. Lucrările sale au fost completate de *Gottlob Frege* (1848 - 1925) și *Alfred Tarski* (1902 - 1983).

Un pas important îl constituie determinarea limitărilor la care ajung logica și calculul computațional. *David Hilbert* (1862 - 1943) a prezentat o listă de 23 de probleme despre care afirma că vor fi subiectul cercetărilor matematicienilor din secolul al XX-lea. Ultima dintre acestea cerea găsirea unui algoritm de determinare a valorii de adevăr a oricărei propoziții care implică numere naturale. *Kurt Gödel* a reușit să demonstreze că în orice limbaj, suficient de expresiv pentru a descrie proprietățile numerelor naturale, există propoziții adevărate a căror valoare de adevăr nu poate fi demonstrată prin nici un algoritm.

Acest rezultat arată că există anumite funcții (pe întregi) care nu pot fi reprezentate de un algoritm, deci nu pot fi calculate. *Alan Turing* (1912 - 1954) a încercat să găsească funcțiile care pot fi calculate. S-a ajuns la concluzia că mașina *Turing* poate calcula orice funcție calculabilă și s-au găsit unele funcții care nu pot fi calculate cu ajutorul acestei mașini.

O noțiune mult mai importantă este cea de ***inaccesibilitate*** a unei probleme. O problemă este considerată inaccesibilă dacă timpul necesar determinării soluției crește exponențial cu mărimea intrării. Deoarece problemele inaccesibile necesită un timp de execuție foarte mare, problema generării comportamentului inteligent trebuie împărțită în subprobleme accesibile. O altă noțiune importantă este cea de reducere; reducerea este o transformare a unei clase de probleme într-o altă clasă de probleme astfel încât soluțiile primei clase să poată fi găsite prin reducerea la anumite probleme din a doua clasă și rezolvarea acestora.

Modul de recunoaștere a unei probleme inaccesibile este dat de teoria *NP-completitudinii* introdusă de *Steven Cook* și *Richard Karp*. S-a reușit demonstrarea NP-completitudinii foarte multor probleme și dacă o problemă NP-completă se reduce la o altă problemă înseamnă că această a doua problemă este NP-completă. Totuși, problema rămâne deschisă deoarece nimeni nu a reușit să demonstreze inaccesibilitatea problemelor NP-complete.

O a treia contribuție a matematicii la dezvoltarea inteligenței artificiale o constituie teoria probabilităților. Italianul *Gerolamo Cardano* (1501 - 1576) a introdus pentru prima dată ideea de probabilitate descriind-o în termeni de prezicere a rezultatelor jocurilor de noroc. Teoria a fost îmbunătățită de mulți alți matematicieni cum ar fi *Pierre Fermat* (1601 - 1665), *Blaise Pascal* (1623 - 1662), *James Bernoulli* (1654 - 1705) și *Pierre Laplace* (1749 - 1827). *Bernoulli* a descris un punct de vedere alternativ asupra probabilităților care a dus la dezvoltarea probabilităților subiective care sunt actualizate pe măsură ce apar noi evenimente. Contribuții în acest domeniu a mai avut și *Thomas Bayes* (1702 - 1761).

Trebuie realizată o legătură între raționamentele probabilistice și acțiuni. Teoria decizională, introdusă de *John von Neumann* și *Oskar Morgenstern*, a fost prima teorie care a reușit să diferențieze acțiunile corecte (bune) de acțiunile incorecte.

Psihologia

Se spune că bazele psihologiei au fost puse de germanul *Herman von Helmboltz* (1821 - 1894) și de studentul său *Wilhelm Wundt* (1832 - 1920). Acesta din urmă a creat laboratoare în care era studiată psihologia folosind metoda introspecției. ***Mișcarea comportamentală***, inițiată de *John Watson* (1878 - 1958) și *Edward Lee Thorndiker* (1874 - 1949), se opune acestei metode subiectiviste și susține că singurele rezultate valabile pot fi obținute pe baza experimentelor.

Punctul de vedere potrivit căruia creierul posedă și procesează informații, care stă la baza ***psihologiei cognitive***, a apărut în lucrările lui *William James* (1842 - 1910). Unul dintre adepții acestuia, *Kenneth Craig*, susținea că scopurile, convingerile și pașii raționamentelor pot fi o parte foarte folositoare pentru studiul comportamentului uman. *Craig* afirma că pentru ca un agent să fie bazat pe cunoaștere trebuie să fie parcurși trei pași. În primul rând, trebuie ca stimulii să fie "traduși" într-o reprezentare internă. În al doilea rând, reprezentarea trebuie să poată fi manipulată prin procese cognitive pentru a obține alte reprezentări interne. În al treilea rând, reprezentările obținute trebuie să fie "traduse" în acțiuni. Acest punct de vedere a dominat psihologia începând cu 1960.

Mult timp nu s-a făcut o distincție clară între IA și psihologia cognitivă. Programele de inteligență artificială erau descrise ca fiind rezultate psihologice fără a se face referiri la comportamentul uman pe care le modelau. În ultimii ani diferențele dintre cele două domenii s-au accentuat și acum o anumită lucrare poate fi clasificată ca fiind parte a unuia dintre ele.

Industria calculatoarelor

Pentru ca inteligența artificială să aibă succes, avem nevoie de două elemente: inteligență și un obiect cu ajutorul căreia să se exprime. Calculatorul a fost unanim acceptat ca fiind obiectul care are cele mai mari șanse să devină inteligent.

Calculatorul digital modern a apărut, aproape simultan, în trei dintre țările implicate în *Al Doilea Război Mondial*. Primul calculator operațional a fost denumit *Heath Robinson* și a fost creat în 1940 de către o echipă condusă de *Alan Turing*. Unicul scop al acestui calculator era descifrarea mesajelor germane. Succesorul său a fost denumit *Colossus* și a apărut în 1943. Până la sfârșitul războiului au fost create zece mașini de acest tip.

Primul calculator programabil a fost *Z-3* și a fost inventat de germanul *Konrad Zuse* în 1941. *Zuse* a inventat reprezentarea în virgulă mobilă și, în 1945, a creat limbajul *Plankalkul*, primul limbaj de nivel înalt.

Primul calculator electronic apărut în SUA a fost *ABC*. El a fost creat între 1940 și 1942 de către *John Atanasoff* și studentul său *Clifford Berry*. Au mai apărut două proiecte secrete al căror scop era construirea unui calculator. La *Harvard* au fost create calculatoarele *Mark* I, II și III de către o echipă condusă de *Howard Aiken*. La Universitatea din *Pennsylvania* a fost creat *ENIAC*, primul calculator care putea fi folosit în scopuri generale. Succesorul său, *EDVAC*, se baza pe ideea lui *John von Neumann* de a folosi un program stocat în memorie.

Probabil că cel mai important pas înainte a fost calculatorul *IBM 701*, construit în 1952 de către *Nathaniel Rochester* și echipa sa. Acesta a fost primul calculator care a adus profituri fabricanților săi.

Fiecare nouă generație de calculatoare a adus îmbunătățiri în ceea ce privește viteza și capacitatea de stocare. Industria calculatoarelor a cunoscut o dezvoltare foarte rapidă, performanțele dublându-se la fiecare doi ani.

Au existat dispozitive pentru efectuarea calculelor cu mult înaintea apariției calculatoarelor. Abacul are o vârstă de peste 7000 de ani. La mijlocul secolului al XVII-lea *Blaise Pascal* a construit o mașină mecanică pentru efectuarea adunărilor și scăderilor. *Leibniz* a îmbunătățit mașina prin adăugarea posibilității efectuării înmulțirilor prin adunări repetate. Peste un secol, *Charles Babbage* a încercat să construiască o mașină care calcula tabelele logaritmice. El nu a reușit să termine proiectul, dar s-a ocupat de ***motorul analitic*** pentru care a introdus ideile de memorie adresabilă, programe stocate în memorie și salturi computaționale. *Ada Lovelace* a scris programe pentru motorul analitic și a susținut că mașina poate fi îmbunătățită pentru a juca șah sau a compune muzică. Ea este considerată a fi prima programatoare din lume. Proiectul lui *Babbage* s-a dovedit viabil în 1993 când a fost construit un model operațional care folosea tehnici mecanice disponibile în timpul lui *Babbage*.

Inteligența artificială folosește și partea *software* a informaticii. Aceasta a oferit sistemele de operare, limbajele de programare și uneltele necesare scrierii programelor moderne. Totuși, și IA a contribuit mult la dezvoltarea acestui domeniu prin introducerea ideii de partajare a timpului, interpretoare interactive, listă înlănțuită, management automat al resurselor. De asemenea, IA a contribuit la introducerea unor concepte cheie cum ar fi programarea orientată obiect și dezvoltarea programelor folosind medii integrate cu interfață grafică.

Lingvistica

În anul 1957 *B.F. Skinner* a publicat cartea *Verbal Behaviour*. Această carte descria abordarea comportamentală a învățării limbajului și era scrisă de cel mai recunoscut expert în domeniu. Totuși, un comentariu asupra acestei lucrări a devenit la fel de celebru ca și cartea. În comentariu, *Noam Chomsky* a arătat că teoria comportamentală nu explică noțiunea de ***creativitate*** în limbaj. Mai exact, ea nu poate explica modul în care un copil reușește să înțeleagă și să formuleze propoziții pe care nu le-a mai auzit niciodată. Teoria lui *Chomsky* a reușit să explice acest fenomen și, spre deosebire de celelalte teorii, era suficient de științifică încât, în principiu, putea fi implementată în programe. Această teorie se baza și pe unele modele sintactice introduse de lingvistul indian *Panini* în jurul anului 350 î.Hr.

Dezvoltările ulterioare ale lingvisticii au demonstrat că problema este mult mai complexă decât părea ea în 1957. Limbajul este ambiguu și multe elemente sunt subînțelese. Pentru a înțelege limbajul trebuie mai întâi să fie înțeles subiectul și contextul discuției și nu doar structura propozițiilor.

Lingvistica modernă și IA s-au "născut" aproape simultan. Din aceste motive lingvistica nu are un rol foarte important în apariția inteligenței artificiale. Totuși, cele două domenii s-au "intersectat" deseori și au dus la apariția unui nou domeniu numit ***lingvistică computațională*** sau ***procesarea limbajului natural***.

Istoria inteligenței artificiale

Vom prezenta acum etapele de dezvoltare ale inteligenței artificiale, prezentând pe scurt fiecare dintre faze.

Nașterea inteligenței artificiale

Prima lucrare care este acum considerată ca fiind un studiu de inteligență artificială a fost scrisă în 1943 de către *Warren McCulloch* și *Walter Pitts*. Ei s-au inspirat din trei surse: cunoașterea fiziologiei de bază și a funcțiilor neuronilor din creier, logica propozițională și teoria computațională a lui *Turing*. Cei doi au propus modelul neuronilor artificiali în care fiecare neuron se poate afla în două stări: activat sau neactivat. Activarea unui neuron are loc doar ca răspuns la o stimulare efectuată de un număr suficient de mare de neuroni învecinați. Starea unui neuron este echivalentă cu valoarea de adevăr a unei propoziții logice. Cei doi au demonstrat faptul că orice funcție calculabilă poate fi evaluată folosind o anumită rețea de neuroni. Tot ei au susținut că ar putea fi proiectată o rețea capabilă de învățare. În 1949, *Donald Hebb* a demonstrat că poate exista o regulă simplă de actualizare pentru modificarea puterii legăturilor între neuroni astfel încât procesul de învățare să poată avea loc.

La începutul anilor '50 au fost scrise primele programe de șah pentru calculatoarele care foloseau arhitectura *von*

Neumann. În 1951 a fost construit primul calculator neuronal numit *SNARC*. Acesta reușea să simuleze o rețea care conținea 40 de neuroni. După 1970 *Marvin Minsky* a demonstrat câteva teoreme care au dus la scăderea interesului pentru cercetarea rețelelor neuronale.

Locul oficial al nașterii inteligenței artificiale este *Daramouth College*. Aici, în 1956, a avut loc un simpozion care a durat două luni și la care au fost invitați numeroși oameni de știință care se ocupau cu teoria automatelor, cu studiul rețelelor neuronale sau cu studiul inteligenței.

Primul program capabil să efectueze raționamente a fost *Logic Theorist*. Programul a reușit să demonstreze o mulțime de teoreme care apăreau într-o lucrare de referință numită *Principia Mathematica*. Pentru una dintre teoreme, programul a găsit o demonstrație mai scurtă decât cea care apărea în lucrarea amintită.

Chiar dacă acest simpozion de la *Daramouth* nu a dus la apariția unor idei revoluționare, el a permis contactul între cei care se ocupau de acest domeniu. În următorii 20 de ani, participanții și studenții lor au fost figurile marcante ale inteligenței artificiale.

Entuziasmul inițial

Primii ani de existență a inteligenței artificiale au fost plini de succese. Datorită faptului că în această perioadă calculatoarele erau destul de primitive iar uneltele de dezvoltare lăsau mult de dorit, era uimitor pentru oricine atunci când vedea că un calculator realizează operații care necesită un anumit grad de inteligență. Anumiți oameni de știință au creat o listă care conținea operațiile pe care, după părerea lor, un calculator nu le va putea realiza niciodată. Cercetătorii din domeniul IA au demonstrat, pe rând, posibilitatea calculatoarelor de a efectua majoritatea acestor operații.

După apariția programului *Logic Theorist* a apărut ***GPS*** (***General Problem Solver***). Acest nou program a fost proiectat în așa fel încât să simuleze raționamentele umane efectuate pentru rezolvarea unei probleme. S-a observat că acest program rezolvă subproblemele care apăreau într-o ordine similară cu cea pe care o folosesc oamenii. Se poate spune că *GPS* a fost primul program care a folosit abordarea gândirii umane a inteligenței artificiale.

Printre primele programe de inteligență artificială s-au numărat și cele realizate la *IBM* de *Nathaniel Rochester* și colegii săi. În 1959 *Herbert Gelenter* a realizat programul ***GTM*** (***Geometry Theorem Prover***). Acesta era capabil să demonstreze teoreme folosind axiome reprezentate explicit.

Începând cu 1952 *Arthur Samuel* a scris o serie de programe pentru jocul de dame. Până la urmă a reușit să creeze un program care să poată concura cu adversari umani. S-a demonstrat faptul că ideea potrivit căreia programele fac doar ceea ce li se "spune" să facă este falsă. Programul creat de *Samuel* a ajuns în scurt timp să joace mai bine decât creatorul său. În februarie 1956 funcționarea programului a fost demonstrată într-o emisiune televizată și a creat o impresie excelentă.

John McCarthy a avut trei contribuții majore la dezvoltarea inteligenței artificiale, toate în anul 1958. El a inventat limbajul de programare *Lisp* care a devenit limbajul de programare cel mai folosit pentru crearea programelor de inteligență artificială. Tot el a avut o contribuție esențială în introducerea partajării timpului. În același an a publicat lucrarea *Programs with Common Sense* în care a descris un program ipotetic numit *Advice Taker* care poate fi considerat primul sistem IA complet. La fel ca și *Logic Theorist* și *Geometry Theorem Prover*, *Advice Taker* folosea cunoștințe pentru a găsi soluții la anumite probleme. Spre deosebire de predecesorii săi, noul program era realizat pentru a lucra cu date ale lumii exterioare și nu cu anumite concepte abstracte. S-a demonstrat că nu este nevoie decât de câteva axiome simple pentru ca programul să genereze un plan care să permită deplasarea la aeroport pentru a ajunge cu suficient timp înaintea plecării avionului. Programul a fost proiectat în așa fel încât să accepte noi axiome care să permită dobândirea de "competențe" în domenii noi fără a mai fi nevoie de o reprogramare. *Advice Taker* înglobează principiile generale de reprezentare a cunoștințelor și efectuare a raționamentelor.

Tot în 1958 *McCarthy* și *Marvin Minsky* au devenit colegi la *MIT*. Timp de câțiva ani cei doi au pus bazele inteligenței artificiale dar, la un moment dat, au apărut divergențe de opinii. *McCarthy* încerca să folosească logica pentru a crea reprezentări ale cunoașterii și raționamente, în timp ce *Minsky* dorea să creeze programe care funcționează și, până la urmă, a introdus un punct de vedere anti-logic. În 1963 *McCarthy* s-a mutat la *Stanford* unde a inițiat un laborator de IA. *J.A. Robinson* a continuat cercetările sale și a reușit să descopere un algoritm complet de demonstrare a teoremelor care implicau logica de ordinul întâi. Cercetările de la *Stanford* s-au concentrat pe metode generale pentru raționamentele logice.

Minsky a coordonat o echipă de studenți care au încercat să rezolve probleme simple care păreau că necesită inteligență pentru a fi rezolvate. Ele se ocupau de domenii limitate care au primit denumirea de ***microlumi***. În 1968 a fost creat programul *Analogy* care reușea să rezolve probleme de analogie geometrică asemănătoare cu cele care apar în testele de inteligență.

Cea mai celebră microlume a fost cea care modela o lume formată din blocuri. Mai multe blocuri de diferite forme (conuri, cuburi, cilindri etc.) și dimensiuni erau așezate pe o suprafață orizontală și un robot trebuia să le aranjeze într-o anumită ordine.

S-a dezvoltat și domeniul rețelelor neuronale. *Winograd* și *Cowan* au arătat în 1963 cum un număr mare de elemente pot fi folosite pentru a reprezenta un concept individual. Cu cât numărul de elemente era mai mare, cu atât creștea robustețea sistemului.

Metodele de învățare au fost dezvoltate de către *Bernie Widrow* și *Frank Rosenblatt*. Ultimul a demonstrat o teoremă celebră, cunoscută sub numele de ***teorema convergenței perceptronilor***.

Revenirea la realitate

Încă de la început, cercetătorii în acest domeniu nu au ezitat să facă predicții referitoare la succesele care vor fi realizate. A devenit celebră o afirmație din 1957 a lui *Herbert Simons*: "*Scopul meu nu este să vă surprind sau să vă șochez - dar cea mai simplă metodă de a rezuma este aceea de a spune că acum există mașini care gândesc, învață și creează. Mai mult, abilitățile lor de a efectua aceste lucruri va crește rapid până când - într-un viitor previzibil - gama problemelor pe care le vor putea rezolva va fi comparabilă cu gama problemelor care pot fi rezolvate de mintea umană*".

Chiar dacă expresia "viitor previzibil" poate fi interpretată în orice fel, unele predicții ale lui *Simons* au fost mai exacte.

În 1958 el a afirmat că în zece ani un calculator va putea deveni campion de șah și că o nouă teoremă matematică importantă va fi demonstrată de către o mașină. Aceste previziuni s-au dovedit a fi mult prea optimiste.

Prima dificultate cu care s-au confruntat programele de inteligență artificială era faptul că nu aveau cunoștințe referitoare la obiectul cu care lucrau. Ele obțineau rezultate doar pe baza unor manipulări sintactice. Programul *Eliza*, creat în 1965 de către *Weizenbaum*, părea că poate purta o conversație despre orice subiect. În realitate, el doar împrumuta și manipula propozițiile scrise de interlocutorul uman. S-a încercat crearea de programe care să realizeze traduceri dintr-o limbă în alta. La început s-a dorit traducerea documentelor rusești apărute cu ocazia lansării satelitului *Sputnik* din 1957. Se credea că, pentru a păstra sensul propozițiilor, sunt suficiente simple transformări sintactice bazate pe gramatica limbii ruse și cea a limbii engleze și înlocuiri de cuvinte bazate pe un dicționar electronic. În realitate, traducerile necesită prezența unor cunoștințe referitoare la subiectul discutat pentru a rezolva ambiguitățile și a stabili conținutul propozițiilor. A rămas celebră traducerea propoziției "*the spirit is willing but the flash is weak*". S-a folosit un program pentru a o traduce în limba rusă și apoi pentru a retraduce expresia obținută în engleză. Rezultatul a stârnit multe zâmbete: "*the vodka is good but the meat is rotten*". În 1966 s-a ajuns la concluzia că nici un program nu a reușit traducerea unui text științific și nici nu sunt speranțe ca să apară un astfel de program în viitorul apropiat.

O a doua dificultate apărută a fost inaccesibilitatea multor probleme pe care IA încerca să le rezolve. Majoritatea primelor programe de inteligență artificială funcționau reprezentând fapte de bază referitoare la probleme și încercând efectuarea unor serii de pași pentru a obține rezultatul. Aceste prime programe furnizau rezultate deoarece microlumile cu care lucrau conțineau foarte puține obiecte. Înainte de apariția teoriei NP-completitudinii se credea că obținerea rezultatelor pentru microlumi mai mari necesita doar mai multă memorie și o viteză crescută de procesare. Au apărut experimente în evoluția mașinilor (actualii algoritmi genetici) care erau bazate pe convingerea (evident corectă) că printr-o serie corectă de mici mutații, aplicate codului sursă al unui program, se poate crea un program care să aibă performanțe satisfăcătoare pentru orice problemă simplă. A apărut și ideea folosirii mutațiilor aleatoare și aplicării unei selecții pentru a păstra mutațiile al căror comportament pare să se îmbunătățească. Nu s-au realizat progrese semnificative în acest domeniu.

O a treia dificultate s-a datorat limitărilor fundamentale a structurilor de bază folosite pentru a obține comportamentul inteligent. De exemplu, un perceptron cu două intrări nu putea să decidă dacă cele două intrări erau identice. Totuși, în 1969 au apărut *algoritmii de învățare cu propagare inversă* care vor fi folosiți pentru renașterea rețelelor neuronale de la sfârșitul anilor '80.

Sisteme bazate pe cunoaștere

Programele pentru rezolvarea problemelor care au apărut la început erau, de fapt, mecanisme cu scop general care încercau să îmbine pași elementari de raționare pentru a găsi soluții complete. Aceste abordări au fost numite metode slabe (*weak methods*) deoarece foloseau puține informații referitoare la subiect. Singura soluție o reprezenta considerarea unor cunoștințe mai potrivite pentru a realiza pași de raționare mai complicați.

Un exemplu al acestor abordări a fost programul *Dendral*. Acest program încerca să determine structuri moleculare pe baza informațiilor furnizate de un spectrometru; acesta a fost primul sistem de cunoaștere intensivă. Următorul pas important a fost făcut în domeniul diagnosticării medicale. A fost creat programul *Mycin* care încerca diagnosticarea infecțiilor de sânge. Folosind aproximativ 450 de reguli, programul a ajuns să aibă performanțe comparabile cu cele ale experților în domeniu.

În această perioadă a devenit evidentă importanța cunoștințelor referitoare la subiect în domeniul recunoașterii limbajului natural. Unii au sugerat că, pentru a crea un program robust pentru recunoașterea limbajului natural, este nevoie de o reprezentare a cunoștințelor referitoare la lume și de o modalitate de a folosi aceste cunoștințe.

În 1973 a fost creat sistemul *Lunar* care permitea geologilor să folosească limba engleză pentru a pune întrebări referitoare la mostrele de roci aduse de pe Lună de misiunile *Appolo*. Acesta a fost primul program de recunoaștere a limbajului natural care a putut fi folosit de o altă persoană în afară de creatorul său. Începând din acest moment, multe programe de acest tip au fost folosite ca interfețe pentru bazele de date.

Au apărut numeroase noi limbaje de programare care puteau fi folosite în domeniul inteligenței artificiale. În Europa s-a răspândit limbajul *Prolog*, în timp ce în SUA se utiliza familia de produse *Planner*.

IA devine o industrie

Primul sistem comercial expert de succes a fost produs la *Digital Equipment Corporation* în 1982. Programul ajuta

la alegerea componentelor *hardware* ale sistemelor comandate de clienți și a permis companiei să economisească 40.000.000$ pe an. În 1988 *DEC* folosea 40 de sisteme expert. Aproape fiecare companie americană avea un grup de angajați care se ocupau cu inteligența artificială și foloseau sau cercetau sisteme expert.

În 1981 japonezii au anunțat proiectul *A cincia generație*. Era un proiect pe 10 ani prin care se încerca construirea unor calculatoare inteligente care să folosească limbajul *Prolog* în același fel în care calculatoarele obișnuite foloseau limbajul mașină. Printre alte scopuri ambițioase ale acestui proiect se număra și cel de a obține înțelegerea completă a limbajului natural.

Acest proiect a impulsionat interesul în inteligența artificială. Cercetătorii au profitat de teama autorităților (care nu doreau ca japonezii să domine domeniul IA) și au primit fonduri pentru proiecte similare. În Statele Unite, compania *MCC* (*Microelectronics and Computer Technology Corporation*) a fost formată pentru a deveni un consorțiu care să contracareze proiectul japonez. În Marea Britanie cercetările în domeniul IA au fost din nou finanțate. În ambele țări, inteligența artificială făcea partea dintr-un larg efort de cercetare care includea proiectarea *chip*-urilor și interfețelor umane.

Au apărut numeroase alte companii care se ocupau de inteligența artificială. Unele ofereau unelte *software* necesare pentru a construi sisteme expert; câteva exemple sunt *Carnegie Group*, *Inference*, *Intellicorp* și *Teknowledge*. Altele ofereau stații de lucru care erau optimizate pentru dezvoltarea programelor scrise în Lisp; câteva dintre ele erau *Lisp Machines Inc.*, *Texas Instruments*, *Symbolics*, *Xerox* etc.

Întoarcerea rețelelor neuronale

Chiar dacă informatica a ignorat o lungă perioadă de timp rețelele neuronale, cercetările asupra acestora au continuat în alte domenii, în special în fizică. Mulțimile mari de neuroni individuali puteau fi privite într-o manieră similară mulțimilor de atomi din solide. În 1982, fizicianul *Hopfield* a folosit tehnici de statistică mecanică pentru a studia proprietățile de stocare și optimizare ale rețelelor. Unii psihologi au continuat studiul memoriei ca rețea de neuroni.

Revitalizarea rețelelor neuronale a avut loc la mijlocul anilor '80 când patru grupuri diferite de cercetători au redescoperit algoritmul de învățare cu propagare inversă, algoritm prezentat pentru prima dată în 1969. Algoritmul a fost aplicat pentru multe probleme de învățare din domeniul informaticii sau psihologiei, ceea ce a dus la o creștere semnificativă a entuziasmului cercetătorilor IA.

În aceeași perioadă de timp au apărut primele deziluzii în ceea ce privește sistemele expert. A devenit clar faptul că prin cumpărarea unui sistem rațional și indicarea unor reguli nu se obține un sistem expert de succes. Mulți au crezut că va apărea o nouă perioadă de stagnare în cercetările asupra inteligenței artificiale. A urmat o perioadă în care rețelele neuronale și IA au fost privite ca domenii rivale și nu ca abordări diferite ale aceluiași subiect.

Evenimente recente

În ultimii ani au avut loc schimbări atât în conținutul cât și în metodologia de cercetare a inteligenței artificiale. Tendința actuală este de a folosi vechile teorii și nu de a introduce altele noi. De asemenea, anumite fapte sunt susținute doar pe baza unor demonstrații teoretice sau a unor experimente riguroase și nu pe baza intuiției. În același timp, se încearcă crearea de programe care să fie relevante pentru lumea reală și nu pentru "lumi de jucărie".

Aceste șabloane au fost ilustrate în cadrul recunoașterii vocale. În anii '70 au fost încercate o mare varietate de arhitecturi și abordări, majoritatea fiind mult prea fragile pentru a funcționa pe mai mult de câteva exemple selecționate. Modelele ascunse *Markov* au revoluționat acest domeniu; ele erau bazate pe rezultate matematice riguroase și permiteau crearea de programe care să învețe prin antrenamente în cadrul cărora se folosea vorbirea curentă a oamenilor. Tehnologia recunoașterii vocale, împreună cu domeniul înrudit al recunoașterii scrisului, sunt suficient de avansate în acest moment pentru a putea fi integrate, în scurt timp, în produsele industriale și de larg consum.

În 1988 *Judea Pearl* a scris cartea *Probabilistic Reasoning in Intelligent Systems* care a dus la acceptarea teoriei probabilităților și a teoriei decizionale în inteligența artificială. Au fost construite modele despre care se putea spune că acționează rațional (potrivit legilor teoriei decizionale) și nu încearcă să imite experții umani.

Alte "revoluții" similare au apărut în robotică, învățarea mașinilor, viziunea asupra calculatoarelor și reprezentarea cunoștințelor. O mai bună înțelegere a problemelor și a proprietăților lor de complexitate a dus la obținerea unor metode de cercetare mai robuste.

Rezultate spectaculoase

În 1989 marele maestru internațional *Arnold Denker* avea în fața sa o tablă de șah și studia piesele. Și-a dat seama că nu mai are nici o șansă și a abandonat. Oponentul său, *Hitech*, a devenit primul program de șah care a reușit să învingă un mare maestru de șah.

În 1994 un călător vorbește la un microfon: "*Vreau să merg de la Boston la San Francisco!*" Răspunsul vine imediat: "*În ce dată doriți să călătoriți?*". Călătorul spune că dorește să plece pe data de 20 octombrie cu o cursă fără escală, să plătească un preț minim și să se întoarcă duminica. Tranzacția este efectuată de *Pegasus*, un program de recunoaștere vocală, și călătorul obține o rezervare pentru zborul dorit. Procedând astfel, el a economisit 894$ față de prețul pe care ar fi trebuit să îl plătească în mod normal. Chiar dacă *Pegasus* nu recunoștea corect decât nouă cuvinte din zece, el putea să corecteze erorile deoarece înțelegea modul în care are loc dialogul.

În 1992, unui analist din sala de operații de la *JPL* (*Jet Propulsion Laboratory*) i-a fost atrasă atenția de către un

mesaj care clipea pe ecran și semnala o anomalie la naveta spațială *Voyager* care se afla în vecinătatea planetei *Neptun*. Din fericire, el reușește să rezolve pe loc problema. Totuși, dacă nu exista sistemul expert *Marvel* care monitoriza transmisiunile efectuate de naveta spațială, această anomalie ar fi trecut neobservată.

În 1993, un șofer călătorește cu 90 km/h pe autostradă și pare foarte relaxat. Pe parcursul ultimilor peste 150 km el nu a mai atins volanul, frâna sau accelerația. Adevăratul șofer este un sistem robotizat care folosește camere video, sonare și lasere pentru a primi informații referitoare la exterior. El combină aceste informații și folosește experiența acumulată în timpul orelor de antrenament pentru a conduce vehiculul.

În 1991 un doctor expert se confruntă cu un caz foarte dificil. Pentru a verifica performanțele unui program de IA el îi prezintă acestuia cazul. În momentul în care calculatorul furnizează diagnosticul medicul surâde cu superioritate. Creatorii sistemului, doar puțin îngrijorați, cer calculatorului să explice modul în care a ajuns la diagnosticul respectiv. Mașina punctează principalii factori care i-au influențat decizia și explică interacțiunile subtile ale unor simptome prezente. Până la urmă expertul și-a recunoscut greșeala.

În 1994, pe un stâlp de iluminat public se afla o cameră de luat vederi folosită pentru a supraveghea traficul dintr-o intersecție pariziană. Dacă un om ar fi citit informațiile de pe un ecran, ar fi văzut propoziții de tipul: "*Citröen 2CV intră pe Champș Elysée din Place de la Concorde*" sau "*Camion cu marca necunoscută staționează în Place de la Concorde*" și așa mai departe. Ocazional, puteau fi citite și mesaje de genul: "*Incident major în Place de la Concorde, coliziune între un motociclist și o camionetă care se deplasa cu viteză*". Simultan cu afișarea unui astfel de mesaj, avea loc și un apel telefonic la serviciile de urgență.

Acestea sunt doar câteva dintre exemplele de sisteme de inteligență artificială care există astăzi. Nu este nici magie, nici SF, ci știință, inginerie și matematică.

La final

În cadrul acestui articol am încercat să prezentăm domeniul de activitate al inteligenței artificiale, precum și modul în care aceasta a apărut. În încheiere vă prezentăm câteva posibile subiecte de discuție.

- Există clase bine-cunoscute de probleme despre care se știe că sunt inaccesibile și alte clase care sunt nedecidabile (nu pot fi rezolvate) pentru orice calculator. Unii spun că aceasta înseamnă că inteligența artificială este imposibilă. Este acest lucru adevărat?
- Să presupunem că un program obține la un test standard de inteligență un IQ de 200 de puncte. Este acest program mai inteligent decât un om? Explicați.
- Dați exemplu de o afirmație a unui expert potrivit căreia până la sfârșitul secolului al XX-lea calculatoarele vor avea o anumită capacitate și care s-a dovedit a fi exagerată.
- Dați exemplu de o afirmație a unui expert potrivit căreia calculatoarele nu vor avea niciodată o anumită capacitate și, în zilele noastre, există deja calculatoare care posedă acea capacitate.
- Se spune că nu există posibilitatea ca un calculator să fie inteligent și că el nu poate efectua decât operațiile pe care i le indică programatorul. Este a doua afirmație adevărată? Adevărul celei de-a doua afirmații implică adevărul primeia?
- Se spune că animalele nu pot fi inteligente și că ele nu pot face decât ceea ce le "spun" genele să facă. Este a doua afirmație adevărată? Adevărul celei de-a doua afirmații implică adevărul primeia?

Răspunsurile la aceste întrebări sunt destul de greu de găsit și oricum, oricare ar fi acestea, vor apărea discuții pe tema lor. Totuși, mulți se joacă încercând să găsească răspunsuri cât mai convingătoare. Încercați și voi! Distracție plăcută!

Bibliografie

Stuart J. Russel, Peter Norvig, *Artificial Intelligence, A Modern Approach*, Editura Prentice Hall, New Jersey, 1995

Mihai Scorțaru este redactor șef adjunct al GInfo. Poate fi contactat prin e-mail la adresa skortzy@xnet.ro.

Microsoft Windows XP

Compania Microsoft a anunțat denumirea viitorului său sistem de operare: Windows XP. Până acum, acesta era cunoscut doar sub numele de cod: Whistler.

Acest succesor al produsului Windows 2000 va introduce o interfață nouă pentru a ușura utilizarea sistemului. Totuși, diferențele nu vor mai fi la fel de semnificative ca cele dintre interfețele Windows 3.x și Windows '9x/Me sau Windows NT/2000. În acest moment, dezvoltarea sistemului se află în stadiul Beta 2.

Noul sistem de operare va combina o serie de caracteristici ale sistemelor de operare actuale Windows Me și Windows 2000.

De asemenea, Windows XP va conține noul program de navigare Internet Explorer 6.0.

PC MAGAZINE ROMÂNIA

COMPUTER PRESS AGORA

www.pcmagazine.ro

Revista
utilizatorilor
de calculatoare personale

COMPUTER PRESS AGORA

Str. Tudor Vladimirescu nr. 63/9, Tg. Mureş (CP 230/1), 4300
Tel. 065-166516, Fax 065-166290, e-mail: office@agora.ro
Str. C-tin Rădulescu Motru nr. 13, etaj 4, apt. 42, sect. 4 (CP 94/49), Bucureşti
Tel: 01-3309282, Fax: 01-3309285

Datorită noului Armada M700 cu procesor Intel® Pentium® III 750 MHz poți acum să faci față oricăror situații neprevăzute. Poți fi în mai multe locuri în același timp și poți face mai multe lucruri deodată. Pentru că Armada M700 are o autonomie de funcționare de peste 5 ore prin folosirea simultană a două baterii iar tastatura Easy Access Internet îți dă posibilitatea conectării imediate la Internet. În plus, cântărește mai puțin de 2,3 kg și are o grosime de numai 3 cm. Astfel că treaba pe care ai început-o la birou o poți termina după program, la cinema. Tehnologia care te inspiră. De la Compaq. Bun venit în noua eră IT.

COMPAQ
Inspiration Technology

EȘTI LA BIROU.
ȘI TOTUȘI NU EȘTI...

COMPAQ ARMADA M700

Procesor Intel® Pentium® III 750MHz • 128MB RAM
• hard disk 12GB • DVD • modem 56k • display TFT 14"
• Ethernet 10/100 • Microsoft Windows 2000/NT4

Calculatoarele Compaq au preinstalat Microsoft® Windows® original.
http://www.microsoft.com/piracy/howtotell

intel inside
pentium® III

scop
COMPUTERS

Barbu Văcărescu 162
Tel: 231 65 78
Fax: 231 65 96
www.scop.ro

© 2001 Compaq Computer Corporation. COMPAQ and the Compaq logo Registered in U.S. Patent and Trademark Office. Armada and Inspiration Technology are trademarks of Compaq Information Technologies Group, LP. in the U.S. and other countries. Intel, the Intel Inside Logo and Pentium are registered trademarks of Intel Corporation.

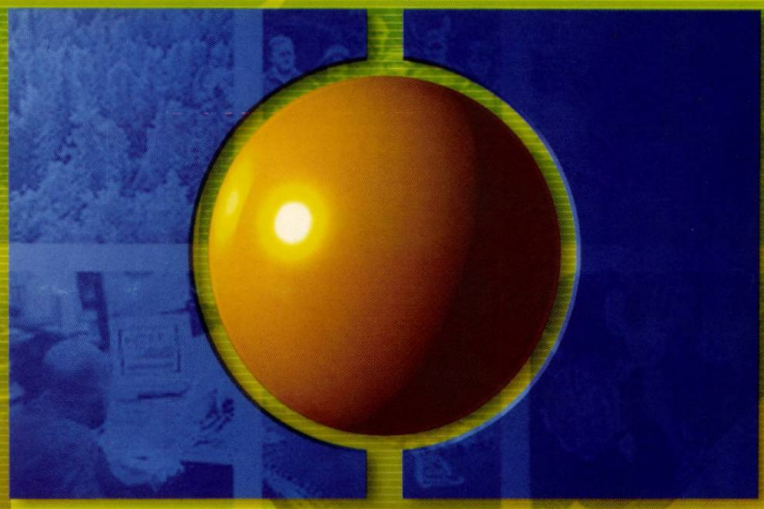
Mai 2001
Vol. 11/5

Lei 15.000

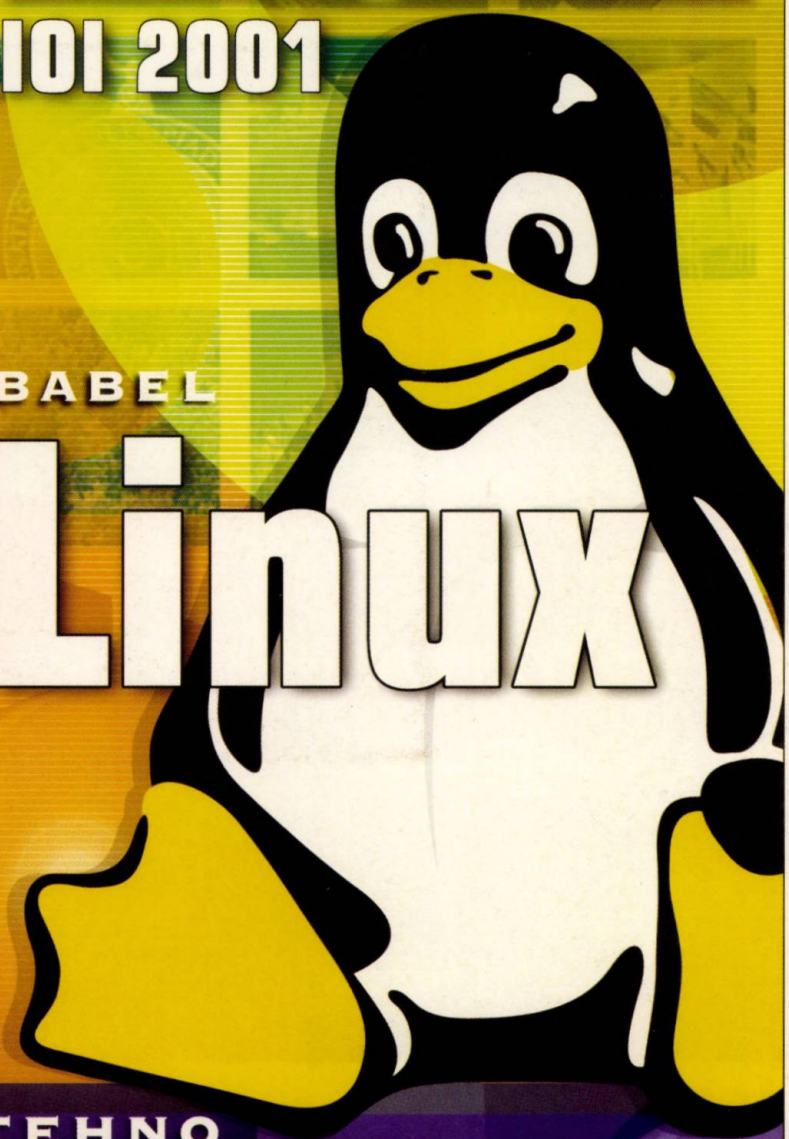
gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

din sumarul numărului:



IOI 2001



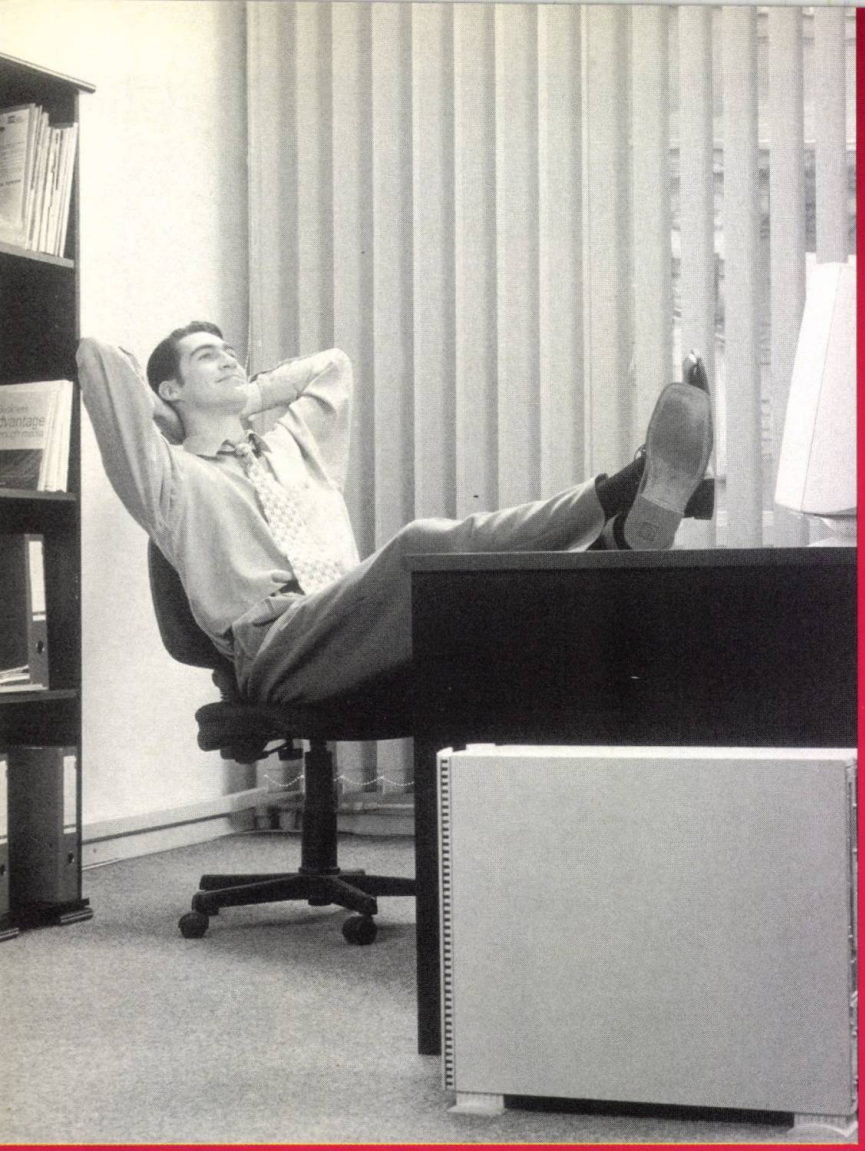
BABEL

Linux

TEHNO

Sisteme de fişiere

Reduceri de preţuri

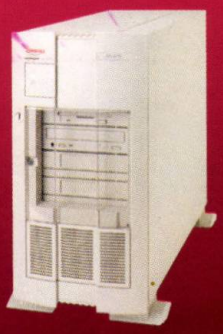


Erai obișnuit ca lucrurile să stea altfel; de acum însă, ai la dispoziție noul server Compaq ProLiant ML 370 cu procesor Intel® Pentium® III 1.0 GHz. Care își vede singur de treabă și își poartă singur de grijă, astfel ca tu să nu mai duci grija lui. Deoarece Compaq Insight Manager și Integrated Remote Console fac posibilă administrarea serverului de la distanță, în timp ce Compaq Availability Agents monitorizează continuu și repară automat erorile obișnuite care pot apărea. Ceea ce înseamnă mai mult timp pentru tine și pentru ideile tale. Tehnologia care te inspiră. De la Compaq. Bun venit în noua eră IT.

COMPAQ
Inspiration Technology

DE ACUM, AI O POZIȚIE DE INVIDIAT.

COMPAQ PROLIANT ML 370
Procesor Intel® Pentium® III 1.0 GHz • 128 MB RAM (max. 4 GB) • hard disk max. 218 GB hot plug • CD Rom • Fast Ethernet 10/100 WOL • adaptor SCSI Ultra Wide 2



intel inside
pentium® III

TREND
IMPORT - EXPORT
www.trend.ro

Str. Popa Savu nr. 77
Tel: 211 01 90
Fax: 211 01 92
sales@trend.ro

© 2001 Compaq Computer Corporation. COMPAQ and the Compaq logo Registered in U.S. Patent and Trademark Office. ProLiant and Inspiration Technology are trademarks of Compaq Information Technologies Group, LP. in the U.S. and other countries. Intel, the Intel Inside Logo and Pentium are registered trademarks of Intel Corporation.

Vol. 11, Nr. 5 /2001

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Redactor:
Carmen Șușman (Alba Iulia)

Au mai colaborat:
Ciprian Ileană (Alba Iulia)
Mihai Pătrașcu (Craiova)
Claudiu Soroiu (Târgu Jiu)

Tehnoredactare:
Mihai Scorțaru

Coperta și grafica:
Adrian Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca

Adresa e-mail: office@ginfo.ro

Tel: 064/192.422
065/166.516
Fax: 065/166.290
http://www.ginfo.ro

ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359

Editor: Computer Press Agora SRL
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihaela Szabo (mszabo@agora.ro)
Ioana Udroiu (iudroiu@agora.ro)

Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

sumar

editorial.. PAG. 3

După ONI și înainte de IOI, CEOI și BOI

Concursuri

concurs.. PAG. 23

Rezultatele rundei a șasea

Clasamentul final

Statistici CPA 2000-2001

actual.. PAG. 5

Olimpiada Națională de INFORMATICĂ

Carmen ȘUȘMAN

babel.. PAG. 31

C și LINUX în Finlanda

Mihai PĂTRAȘCU

Regulament IOI 2001

probleme.. PAG. 9

propuse pentru REZOLVARE

Olimpiada Națională de Informatică

~Bacău, aprilie 2001~

tehno.. PAG. 38

REDUCERI de prețuri

Mihai SCORȚARU

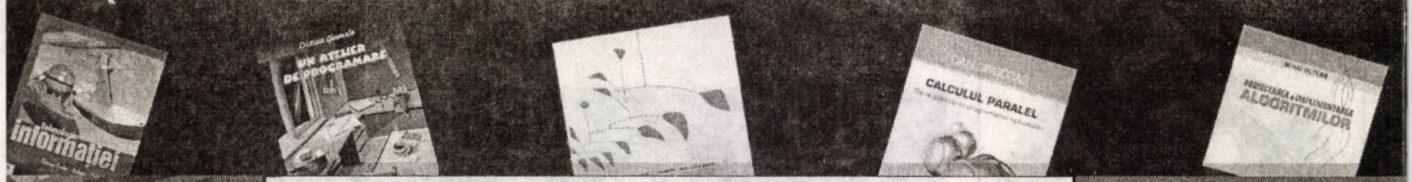
Sisteme de FIȘIERE

Ciprian ILEANĂ
Claudiu SOROIU

www.ginfo.ro

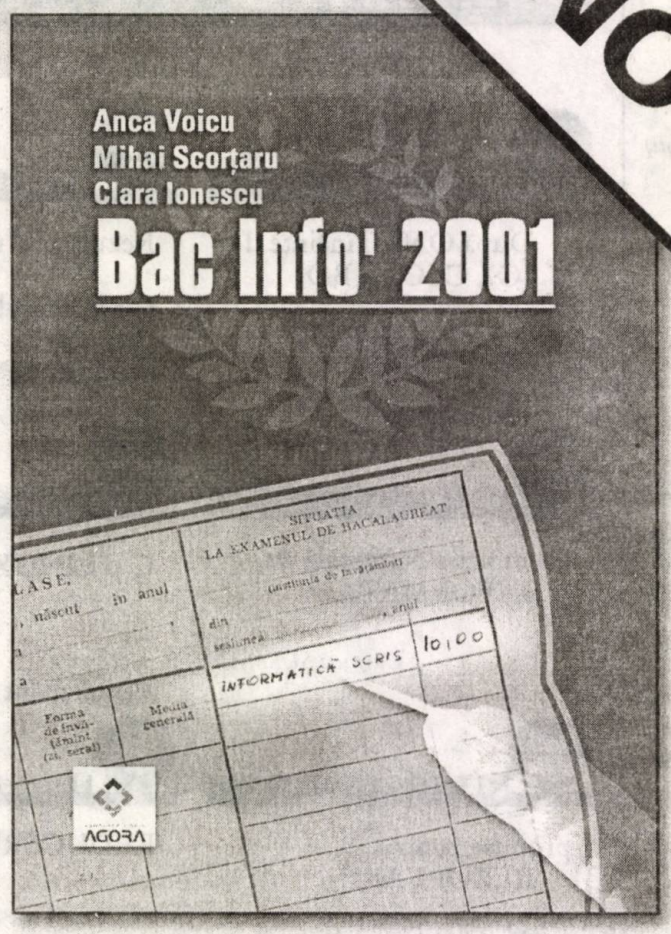
1 octombrie 2001
lansarea noului site

Secțiuni: probleme propuse
soluții software hardware
Concursul de Programare Agora



COMPUTER LIBRIS AGORA

NOUTATE



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

După ONI și înainte de IOI, CEOI și BOI

Evenimentul anului, pentru mulți tineri pasionați de informatică îl constituie Olimpiada Națională de Informatică. Dar știm cu toții că mai urmează câteva "evenimente" care cu siguranță sunt urmărite cu interes de cititorii revistei noastre. Ca în fiecare an, și în 2001, se vor organiza Olimpiada Internațională de Informatică (în Finlanda), Olimpiada de Informatică a Europei Centrale (în Ungaria) și Olimpiada Balcanică de Informatică (în Albania).

Cea de-a XXVI-a ediție a *Olimpiadei Naționale de Informatică* s-a desfășurat anul acesta între 17 și 24 aprilie, la Bacău. Fiecare ediție aduce noutăți, experiența anilor trecuți permite îmbunătățirea anumitor mecanisme, organizatorii locali caută și găsesc soluții noi. Dar soluțiile noi generează probleme noi, care la rândul lor, și ele trebuie rezolvate.

La Bacău am beneficiat toți - concurenții și membrii comisiei - de o organizare foarte bună care a dovedit mult devotament, dăruire și multă muncă efectuată cu entuziasm de către gazde.

Inspectoratul Școlar Județean Bacău și-a făcut simțită prezența și interesul pentru această olimpiadă prin munca susținută a domnului inspector școlar general ***prof. Gheorghi Iorga***, a doamnei inspector general adjunct ***prof. dr. Livia Sibiștean*** și a domnului inspector de matematică-informatică ***prof. Costică Lupu***.

Presupunem că secretul reușitei trebuie căutat și în colaborarea eficientă dintre inspectorat și prefectură, prin bunăvoința domnului prefect ***Cătălin Mardare*** și *Colegiul Național Ferdinand I*, unde doamna directoare ***prof. Antoneta Pătrașcu*** a pus la dispoziția olimpiadei toată școala, toate calculatoarele și întreaga suflare din colegiu.

Profesorii din catedra de informatică (***prof. Ilie Niță***, ***Mircea Ambrozie*** și ***prof. Bogdan Pătruț***) au fost prezenți din prima, până în ultima clipă și au fost prompți la orice solicitare.

Nu putem să nu mulțumim din suflet doamnelor profesoare ***Cristina Sichim*** și ***Ana Întuneric*** pentru modul rapid, "ușor" și înțelegător cu care au ajutat comisia și olimpiada în general, în rezolvarea oricărei probleme care a apărut pe parcurs.

Pe durata desfășurării olimpiadei, gazdele au organizat o expoziție cu vânzare deosebit de reușită unde doritorii au putut vedea și cumpăra cărți, componente *hardware* și *software*.

Comisia Centrală a olimpiadei a fost, conform tradiției de câțiva ani, formată din profesori (universitari și de liceu) și din studenți, foști olimpici. Doamna directoare Pătrașcu a remarcat, printre altele, că la nici o disciplină nu a văzut o comisie atât de "tânără". Specificul olimpiadelor de informatică, pe de o parte ***impune***, pe de alta ***permite*** prezența studenților în comisie. Pe parcursul anilor s-au pus o serie de întrebări în legătură cu această prezență.

Avem convingerea că acești studenți lucrează cu responsabilitate, munca lor este deosebit de valoroasă și micile neajunsuri trebuie puse doar pe seama lipsei de maturitate. Rămâne să îi sfătuim - parafrazându-l pe regretatul profesor ***Grigore C. Moisil*** - *să se străduiască zilnic să-și repare defectul de a fi prea tineri.*

Revenind la bilanțul care se face după fiecare olimpiadă, nu putem să nu observăm că problemele au fost destul de grele, poate mult prea grele pentru participanții la această ediție.

Să fie de vină faptul că în România, de doi ani s-a redus numărul de ore de informatică în licee? Nu prea credem: rezultatele cele mai echilibrate le-au obținut concurenții de clasa a IX-a.

Rămâne să căutăm alte explicații, dar până le găsim, remarcăm faptul că din cele două rele: probleme prea ușoare sau prea grele, o considerăm mai bună pe cea de-a doua variantă.

Am fi preferat ca la baraje, rezultatele să fie mai convingătoare. Clasamentul începe cu rezultatul de excepție al lui ***Mihai Pătrașcu*** (560 de puncte din 600 posibile), dar următorul punctaj este de doar 366 de puncte (aparține lui ***Mugurel Andreica***). Ne vom întreba mult timp ce s-a întâmplat cu ***Daniel Dumitran*** care după medalia de aur de la *IOI 2000*, acum s-a clasat doar pe locul 18.

Ca o noutate în ceea ce privește pregătirea problemelor de concurs, cititorii revistei noastre au aflat din numărul precedent al ***GInfo***, că am

adunat un număr considerabil (56) de probleme cu o lună înainte de începerea olimpiadei. Nu putem afirma că acest număr mare de enunțuri ar fi constituit un avantaj, deoarece în aceste condiții, în timp real, la Bacău, au apărut prea puține probleme "noi", să le numim de rezervă, care ar fi avut menirea să ne sprijine în diversificarea problemelor în cele două zile de concurs. Faptul că am cunoscut enunțurile din timp a permis definitivarea lor în condiții liniștite și nu în timpul zilelor de concurs, unde ne-am putut concentra asupra altor obligații, cum ar fi, de exemplu, evaluarea.

Am lăsat la urmă bucuria de a constata că în condițiile în care enunțurile au fost cunoscute de mai multe persoane din comisie cu mult timp înainte de olimpiadă, nu s-a produs nici o "scurgere de informații". Întâmplarea din jurul problemei "Alpinistul" de la clasa a X-a (au fost două probleme denumite identic și cu un conținut aproape identic pe două liste de dialog la care au fost și sunt înscriși mulți dintre participanții la ONI) nu poate fi catalogată altfel decât "o întâmplare".

Despre procedura de evaluare și *software*-ul propriu-zis de evaluare s-a discutat destul de mult. Unii au încredere în acesta, alții nu. Important este că reacțiile concurenților, conținutul contestațiilor (relativ reduse) au demonstrat faptul că participanții la olimpiadă s-au obișnuit cu acest tip de evaluare și au încredere în rezultatele furnizate de acesta.

Este regretabil că cei care învinuiesc evaluatorul nu știu că acesta lucrează cu programe de verificare și cu fișiere pregătite de membrii comisiei. Dacă s-au strecurat greșeli, nu sistemul de evaluare a fost de vină, ci aceste componente ale sale. De asemenea, ne pare rău că solicitarea comisiei de a avea pentru evaluare "mașini DOS" nu a fost suficient de clară, astfel pe cele câteva calculatoare pregătite în acest sens procesul de evaluare a fost lent. Soluția ideală o constituie realizarea concursului în rețea (dezactivată pe durata probelor) care ar trebui activată doar după plecarea concurenților din sălile de concurs. Astfel, fiecare lucrare s-ar evalua pe "propriul" calculator și în paralel. Dacă am folosi sistemul de operare *Linux*, aceste probleme s-ar rezolva de la sine.

Pentru a asigura o transparență și o fiabilitate mai ridicată, am dori ca anul viitor să lucrăm cu două sisteme de evaluare, în paralel. Greșelile noastre nu vor fi descoperite mai repede, dar vom avea un control mai mare.

A fi membru într-o asemenea comisie este o responsabilitate și înseamnă mult mai multă muncă decât se poate aprecia din afară. Anul acesta apelul de a trimite probleme pentru ONI a ajuns (sperăm) la toți profesorii de informatică din țară. Totuși s-au înscris doar patru profesori "noi" în lista propunătorilor. Să fie pur și simplu o inexplicabilă apatie, sau mai degrabă neîncredere în reușita de a "pătrunde" printre oamenii vechi? Indiferent care este motivul, tendința de a da probleme identice în toată țara la olimpiadele județene nu ni se pare o idee fericită. Dacă subiectele vor "veni de la Minister" oare când va mai avea un profesor de liceu posibilitatea, sau mai bine zis, obligația de a compune o problemă originală? Pe de altă parte o asemenea etapă ar asigura o selecție pentru participarea la națională după valoarea lucrărilor. Oricum, aceste subiecte nu vor fi obligatorii, deci județele vor avea posibilitatea să-și păstreze "independența" și "suveranitatea".

Olimpiada Națională și în acest an a avut sponsori locali și naționali. Cinste și mulțumiri celor care au venit și fie au sprijinit organizarea, fie au adus premii remarcabile premianților.

Ar fi extraordinar dacă, în viitor, numărul lor ar crește, prin aceasta asigurând tinerii talentați și capabili de performanțe, că îi așteaptă un viitor chiar și în România...

Concurenții *Olimpiadei Internaționale de Informatică* din Tampere (Finlanda) vor avea parte de o serie de surprize. Nu vrem să intrăm în speculații privind natura acestora (surprize plăcute sau neplăcute?), dar un lucru este sigur: organizatorii vor schimba sistemul de operare.

În primul rând, se va lucra sub *Linux*, permițând totodată celor care "nu au auzit despre acesta", să lucreze sub *Windows '98 SE*. Mulți consideră că Finlanda este locul ideal pentru această schimbare, datorită și faptului că ***Linus Torwald***, creatorul *Linux*-ului, este finlandez, dar nu este vorba doar de atât.

Linux/Unix sunt sisteme portabile, fiabile și prin compilatoarele dezvoltate pentru aceste platforme, asigură o serie de facilități (executabile mai rapide, memorie - teoretic - oricât de mare) de care un programator se va bucura cu siguranță.

Din păcate, membrii lotului nu vor avea suficient timp să "învețe" să beneficieze de toate funcțiile din biblioteca gcc, dar sperăm că majoritatea se vor descurca, dacă nu în *C*, atunci în *freepascal*. Este o lecție nouă pe care trebuie s-o învățăm, dacă nu am învățat-o până acum.

Organizatorii promit instalări standard ale celor două sisteme de operare de la care se vor abate doar dacă va fi absolut necesar, respectiv în funcție de *hardware*-ul pe care îl vor avea, de compatibilitățile dintre cele două platforme (*Linux* și *Windows*) mai ales că doresc să pună la dispoziția participanților aceleași compilatoare.

Mediul *Windows* reprezintă un pericol din punct de vedere al securității, deci va fi necesar să fie configurat astfel încât performanțele sale să devină similare cu cele ale *Linux*-ului. S-ar putea ca și mărimea acestui *software* să ridice anumite probleme. Este clar că nu va fi nevoie de o instalare completă a *Windows*-ului.

Detalii suplimentare referitoare la *Olimpiada Internațională de Informatică* din acest an veți putea găsi în articolul de la pagina 35.

În încheiere, urăm tuturor cititorilor *GInfo* "Vacanță plăcută!".

Clara Ionescu, Horia Georgescu

Olimpiada Națională de INFORMATICĂ

Carmen Șușman

Anul acesta ONI a avut loc la Bacău și a fost găzduită de Colegiul Ferdinand I. La olimpiadă au participat elevi de liceu din toate județele țării, din Municipiul București și din Republica Moldova. Ca de obicei, numărul concurenților a fost relativ mare (peste 250), fapt care anunța "o bătălie pe cinste" și un concurs frumos. Datorită faptului că m-am aflat la fața locului, îmi permit să vă relatez pe scurt modul în care s-a desfășurat olimpiada de la Festivitatea de Deschidere până în momentul afișării listei cu cei 15 (de fapt 16) componenți ai Lotului Național de Informatică.

La început...

Olimpiada Națională de Informatică din acest an s-a desfășurat în perioada 17-24 aprilie la *Bacău*. Vechiul oraș bacovian a fost încântător și gazdele primitoare, amabile și au comunicat cu noi în speranța ca totul să se desfășoare în condiții optime, atât pentru membrii comisiei cât și pentru participanți. Cred că au reușit acest lucru. Cel puțin s-au respectat cerințele venite de la Minister. Organizatorilor li s-a cerut ca stațiile de lucru să folosească sistemul de operare *Windows 95/98* și să fie instalate pachetele *Borland C++ 3.1* și *Borland Pascal 7.0*. În plus, pentru a ușura munca participanților și a comisiei s-a cerut și instalarea utilitarului *Windows Commander*. Totul ar fi fost perfect dacă programul de evaluare nu ar fi fost incompatibil cu sistemele de operare *Windows*. Până la urmă, totul s-a rezolvat și configurațiile software ale calculatoarelor au fost modificate.

Cele opt săli de concurs și sălile subcomisiilor au fost pregătite până la urmă pentru "atac". Programul era stabilit până în cele mai mici detalii și distribuit tuturor participanților. Toate locurile de cazare ale elevilor și profesorilor se aflau la distanțe relativ mici față de locul desfășurării concursului. Acest lucru a mulțumit pe multă lume: "*Este excelent că internatul se află foarte aproape de liceu. Anul trecut, la Constanța, aveam prea mult de mers până la liceul unde se desfășurau probele.*" declară ***Dan Popovici***, elev în clasa a XI-a la *Liceul "Moise Nicoară"* din *Arad*.

Încă din prima zi localnicii au făcut o impresie cât se poate de plăcută tuturor.

Festivitatea de deschidere

În dimineața zilei de 18 aprilie, la *Ateneu*, s-a anunțat oficial deschiderea olimpiadei. Spre marea surprindere a tuturor, a avut loc o festivitate scurtă în cadrul căreia au luat cuvântul președintele *Comisiei Centrale* ***dl. prof. univ. dr. Horia Georgescu***, ***d-na. prof. Rodica Pintea***, reprezentantă a *Ministerul Educației și Cercetării*, oficialități ale județului *Bacău* și reprezentanți ai *Colegiului Ferdinand I*. Tot atunci au fost prezentați distinșii membri ai comisiei, care au fost întâmpinați cu ropote de aplauze. După festivitate, în holul *Ateneului* am auzit un elev spunând "*Toate momentele frumoase se termină repede*".

Cele două zile de concurs

În zilele de 19 și 20 aprilie a avut loc concursul propriu-zis. Datorită faptului că nu erau destule calculatoare disponibile, probele au avut loc în două etape: clasele a XI-a și a XII-a în cursul dimineții, iar clasele a IX-a și a X-a după-amiază.

Toți concurenții au avut emoții mari, dar destul de mulți au reușit să le stăpânească. Rezultatele ar fi trebuit afișate după fiecare zi de concurs, dar, din nefericire, au existat întârzieri. Unii au fost foarte entuziasmați de punctajele obținute, însă, ca la orice concurs au existat și nemulțumiți. Aceștia din urmă au depus contestații, dar reevaluarea problemelor menționate de ei nu a adus decât puține modificări ale punctajelor. Majoritatea contestațiile nu au fost fondate, însă au existat și excepții, deoarece câțiva dintre concurenți au obținut chiar și

maximul de puncte care revenea problemei respective.

O parte dintre concurenți au pierdut puncte prețioase din pricina neatenției și datorită faptului că nu au ascultat indicațiile date de membrii comisiei. În rest, probele au decurs normal; a existat doar o mică problemă cu un calculator năstrușnic care refuza să funcționeze corect. Din fericire acest incident a fost soluționat urgent. Pentru a veni în ajutorul elevilor, membrii comisiei au stabilit durata probelor în așa fel încât fiecare concurent să poată avea timp pentru citirea și înțelegerea problemelor, pentru întrebări și rezolvări.

Erorile concurenților au fost foarte asemănătoare cu cele ale participanților la *Concursul de Programare Agora*. Acest lucru ne surprinde cu atât mai mult cu cât am observat că mulți dintre concurenții de la *ONI 2001*, au participat și la concursul organizat de revista noastră și noi am semnalat deseori cele mai frecvente erori. Voi aminti doar câteva dintre acestea: neinițializarea variabilelor locale sau a variabilelor dinamice, scrierea incorectă a numelor fișierelor de intrare sau de ieșire, folosirea unit-ului `CRT`, încheierea programelor printr-o instrucțiune de tip `Readln` etc.

Ultima zi de concurs s-a încheiat noaptea târziu, toți participanții au sărbătorit în discoteca *XXL*. A fost o atmosferă plăcută, muzică, dans, dar dezamăgirea cea mai mare a tuturor a fost lipsa fetelor... Trebuie să menționez că la această olimpiadă au fost prezente doar 11 fete! Chiar dacă au fost puține, câteva dintre ele au luat premii. ***Alexandra Constantin*** de la *Liceul "Ion Luca Caragiale"* din *Ploiești* care a obținut o mențiune și a fost cea mai aplaudată fată. De asemenea, mai avea un pas (24 de puncte) și intra în *Lotul Național*.

Ziua de "odihnă"

După probele extrem de obositoare de concurs toată lumea avea nevoie de o pauză... A fost organizată o excursie în jurul *Bacăului*.

Doritorii au urmat un traseu pe ruta *Bacău - Onești - Târgu Ocna - Slănic Moldova - Comănești - Moinești - Bacău*.

Unii au preferat să rămână în *Bacău* pentru a afla cât mai repede rezultatele contestațiilor. Datorită acestora, membrii comisiei nu s-au putut bucura de plăcerile excursiei. Au trebuit reevaluate o mulțime de soluții ale concurenților, multe dintre acestea conținând erori bizare. Astfel, nici unul dintre membrii comisiei nu a putut vizita *Casa Memorială George Enescu* din *Tescani* sau alte puncte interesante ale traseului. Sper că cel puțin unii dintre concurenți nu au regretat că au preferat să meargă în excursie.

Festivitatea de premiere

După ce au rezolvat toate contestațiile, membrii comisiei au decis care dintre participanți vor fi premiați. Numele acestora au fost citite la festivitatea de premiere. Cei mai buni tineri informaticieni au primit diplome și premii din partea *Ministerului Educației și Cercetării* și din partea sponsorilor. Până anul trecut, premiile acordate de Minister erau simbolice; anul acesta ele au crescut de 10 ori. Din păcate, premiile acordate de sponsori au fost mult mai mici decât în anii precedenți.

Datorită numărului mare de premianți, această festivitate a durat mult mai mult decât cea de deschidere, dar, evident, a fost mult mai interesantă. Au vorbit, din nou, reprezentanți ai comisiei, ai gazdelor și ai oficialităților locale. Cel mai scurt discurs a fost al domnului ***Horia Georgescu*** care ar putea primi un premiu pentru cele câteva secunde în care a vorbit. Totuși, acesta a fost și cel mai aplaudat "discurs".

După premiere, cei care nu au obținut distincții la această olimpiadă au părăsit Bacăul; ceilalți au rămas pentru proba de baraj în urma căreia s-a selecționat *Lotul Național de Informatică*.

În seara zilei, membrii comisiei, profesorii însoțitori și gazdele au participat la o masă festivă. Știu că acum câțiva ani la masa festivă participau și concurenții, dar...

Barajul

Probele de baraj au avut loc în 23 și 24 aprilie. Problemele au fost foarte dificile și au existat foarte puțini concurenți care au reușit să obțină punctajul maxim cel puțin la una dintre ele.

Din păcate, aici au apărut cele mai mari "disfuncționalități". Problema ***Deșert*** din prima zi a trebuit reevaluată de două ori deoarece datele de test folosite pentru evaluare nu respectau specificațiile din enunțurile primite de concurenți.

După corectare și rezolvarea puținelor contestații au fost selectați membrii *Lotului Național de Informatică*. Doi concurenți din *Republica Moldova* au obținut punctaje care le-ar fi permis să devină membri ai lotului. Surprinzător, ***Daniel Dumitran***, medaliat cu aur la ultima ediție a *Olimpiadei Internaționale de Informatică*, nu a reușit să se claseze între primii 15. Totuși, membrii comisiei au hotărât includerea sa în lot datorită rezultatelor obținute anterior.

Evaluarea

Pentru evaluarea soluțiilor furnizate de concurenți la *ONI 2001*, a fost adaptat sistemul de evaluare de la *CEOI 2000*. Anul trecut, la Cluj, evaluarea tuturor problemelor dintr-o anumită zi de concurs dura aproximativ 30 de minute deoarece testarea avea loc simultan, pe toate calculatoarele. La *Bacău*, nu a existat această posibilitate datorită faptului că nu era posibilă utilizarea rețelei din liceu în acest scop. Din acest motiv, sursele concurenților au fost "transportate" la comisiile de corectare unde au fost evaluate. Totuși, aceasta nu explică întârzierile exagerate cu care au fost anunțate oficial rezultatele. Principala cauză a acestor întârzieri a fost aceea că programele de verificare folosite de sistemul de evaluare conțineau erori.

Concluzii

Din păcate, a fost evident faptul că anul acesta majoritatea concurenților nu au fost la fel de bine pregătiți ca în anii trecuți. Cu excepția clasei a IX-a la celelalte clase doar câțiva concurenți au obținut punctaje mari. S-a

ajuns în situația în care au fost acordate mențiuni unor concurenți care au obținut mai puțin de 100 de puncte din 600 posibile.

Situația a fost aceeași la *Baraj*. Primul clasat a obținut 560 de puncte, dar pentru a deveni membru al lotului erau suficiente 170 de puncte (punctajul ultimului concurent admis în lot). Au urmat șase participanți cu 162 de puncte, printre care și *Daniel Dumitran*.

Totuși, ceilalți cinci nu au putut fi admiși în lot deoarece resursele financiare ale *Ministerului Educației și Cercetării* nu permit formarea unui lot cu 21 de membri.

Până la urmă, putem considera că această ediție a *Olimpiadei Naționale de Informatică* a fost un succes, în ciuda micilor dificultăți și disfuncționalități care au apărut.

Nu pot încheia fără a-mi exprima speranța că membrii lotului vor avea parte de o pregătire eficientă, iar la olimpiadele internaționale vor participa cei mai buni dintre ei.

Carmen Șușman este redactor al GInfo. Poate fi contactată prin e-mail la **schummy@email.ro**.

Premianții

Clasa a IX-a

Premiul I

Răzvan Dinu, Prahova
Constantin Jucovschi, Rep. Moldova
Cristian Custocheraș, Dâmbovița
Alexandru Moșoi, București

Premiul II

Dan Ghinea, București
Silviu Julean, Arad
Alexandru Preoteasa, București
Andrei Ismail, Galați
Emilian Miron, Bacău
Dan Dragomir, Alba

Premiul III

Horia Ciochină, Iași
Daniel Marinescu, Constanța
Lehel Gazdag, Covasna
Adrian Cearnău, București
Andrei Matei, București
Andrei Homescu, Gorj
Tiberiu Chiorean, Maramureș
Flaviu Roman, Cluj
Andrei Blaj, Suceava
Alexandru Mihăiuc, Vrancea
Constantin Dumitrașcu, Vrancea

Clasa a X-a

Premiul I

Radu Berinde, București
Arcadie Crăcan, Iași

Premiul II

Ștefan Ciobacă, Suceava
Victor Costan, București
Victor Vernescu, Constanța
Dănuț Enăchioiu, Argeș
Tiberiu Dăneț, București
Marius Mitea, Vrancea

Premiul III

Marian Raica, Gorj
Aurel Ghioca, Constanța
Lucian-Daniel Stanciu, Brăila
Valentin Trufin, Suceava
Claudiu Matei, București
Flaviu Pașca, Bistrița
Adrian Airinei, Bacău
Mihai Cristian, Botoșani
Takács Gábor, Bihor

Clasa a XI-a

Premiul I

Daniel Dumitran, București
Dumitru Codreanu, Rep. Moldova
Cristian Băicoianu, Prahova

Premiul II

Vlad Dascălu, Bacău
Andrei Benea, Vrancea
Andrei Markovits, Satu Mare
Dan Popovici, Arad
Vlad Radu, Caraș Severin
Bors Vencel István, Bihor
Florin Buga, Olt
Bogdan Nicolae, Sibiu

Premiul III

Vlad Bologa, Bihor
Paul Rusu, Cluj
Mihai Cosma, Bistrița
Tiberiu Tesileanu, București
Adrian Balasko, Sălaj
Șerban Alexe, Buzău
Bogdan Harjoc, Alba
Maximilian Machedon, București

Clasa a XII-a

Premiul I

Mihai Pătrașcu, Dolj
Mugurel Andreica, București
Pătcaș Csaba, Bihor
Constantin Marina, Alba
Cosmin Negruseri, Bistrița
Daniel Crișan, Vâlcea

Premiul II

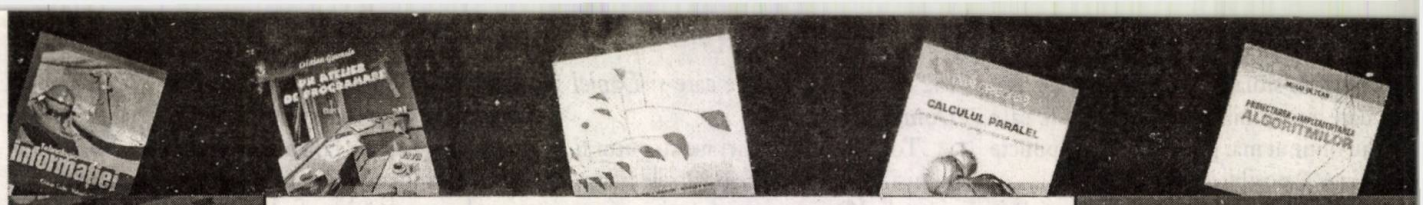
Cristian Alexandrescu, Botoșani
Cătălin Alexandru, Botoșani
George Drumea, București
Ionel Pop, Sălaj
Andrei Gheorghe, București
Robert Bădiță, Dolj

Premiul III

András Csaba, Bihor
Bogdan Piloga, Timiș
Andrei Adler, București
Mihnea Baltă, Constanța
Andrei Toma, Brașov
Codruț-Lucian Lazăr, Alba
Ionuț Trifan, Buzău
Dumitru Daniliuc, Rep. Moldova
Dorin Pena, Prahova
Daniel Ioniță, București
Florin Anghelache-Ionescu, Prahova
Bogdan Stan, Prahova
Radu Timofte, Iași
Florin Ghețu, Vrancea
Claudiu Filimon, București
Alexandru Badea, Constanța
Bogdan Lucaciu, Mehedinți

Lotul Național de Informatică

Mihai Pătrașcu, Dolj
Mugurel Andreica, București
András Csaba, Bihor
Constantin Marina, Alba
Mihai Pantiș, Cluj
Vlad Dascălu, Bacău
Florin Ghețu, Vrancea
Victor Costan, București
Aurel Ghioca, Constanța
Cătalin Alexandru, București
Bors Vencel István, Bihor
Daniel Ioniță, București
Andrei Gheorghe, București
Răzvan Dinu, Prahova
Robert Bădiță, Dolj
Daniel Dumitran, București



COMPUTER LIBRIS AGORA

"Cum aș putea să justific existența acestei cărți? Cum aș putea spre exemplu să justfic cele 100 de pagini despre programarea dinamică, acum când această metodă nu mai figurează în programele școlare? Nu știu. Sau, cum aș putea să justific capitolul despre algoritmi genetici, când pe plan mondial abia de câțiva ani a fost dovedită utilitatea și eficiența lor? Din nou nu știu! Dar rămâne bănuiala: a scrie algoritmi înseamnă a crea. Și nu poți crea fără să gândești."

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

PROBLEME propuse pentru REZOLVARE

În perioada 17-24 aprilie a avut loc, la Bacău, Olimpiada Națională de Informatică. În cadrul acestui număr vă prezentăm enunțurile problemelor propuse spre rezolvare în cele patru zile de concurs.

Clasa a IX-a

P050101: Ferma

prof. Maria Niță și prof. Adrian Niță, Oradea

Un fermier are un teren care are forma unui tablou dreptunghiular lung de *M* unități și lat de *N* unități. Pe teren sunt plantați din loc în loc copaci, pe care fermierul nu dorește să-i taie. Dorind să-și supravegheze cultura, fermierul realizează un mic robot de formă pătrată având latura de trei unități pe care îl poate teleghida prin fermă, parcurgând unitate cu unitate o anumită suprafață.

Robotul se poate mișca pe verticală și pe orizontală dar, nu poate trece peste copaci, nu îi poate distruge, nu se poate roti și are nevoie pentru mișcare de o suprafață corespunzătoare dimensiunii lui.

Cerință

Ajutați-l pe fermier să determine suprafața maximă pe care o poate urmări, folosind acest sistem.

Date de intrare

Fișier de intrare: `FERMA.IN`

Linia 1: `N M`

- două numere naturale nenule, separate printr-un spațiu, reprezentând numărul de linii (*N*), respectiv numărul de coloane (*M*);

Liniile 2...*N*+1: $C_1C_2 \ldots C_M$

- aceste *N* linii codifică ferma; fiecare linie conține câte *M* caractere (fără să fie separate prin spații) cu semnificația:
  - `'.'` - teren liber;
  - `'+'` - locul în care este plantat un copac;
  - `'R'` - centrul robotului.

Date de ieșire

Fișier de ieșire: `FERMA.OUT`

Liniile 1...*N*: $C_1C_2 \ldots C_M$

- aceste *N* linii codifică modul în care fermierul poate să-și utilizeze robotul pe terenul său; fiecare linie conține câte *M* caractere (fără să fie separate prin spații) având semnificația:
  - `'.'` - teren neacoperit de robot;
  - `'*'` - teren ce poate fi verificat de robot;
  - `'+'` - loc în care a rămas copacul.

Restricție

- `1 ≤ N, M ≤ 50`

Exemplu

FERMA.IN	FERMA.OUT
12 11	....*****..
.....	...+*****+.
...+.....+.	..*********
.....	..*********
.....	.+*********
.+.....	...+*******
...+.....	.+.********
.+...R.....	...******+.
.....+.	..+******.+
..+.....+	******+....
......+....	******.....
.....	******+....
......+....	

Timp de execuție: 3 secunde/test

P050102: Fracții

prof. Ovidiu Domșa, Alba Iulia

O proprietate interesantă a fracțiilor ireductibile este că orice fracție se poate obține după următoarele reguli:

- pe primul nivel se află fracția `1/1`;
- pe al doilea nivel, în stânga fracției `1/1` de pe primul nivel, plasăm fracția `1/2` iar în dreapta ei fracția `2/1`;

nivelul 1: 1/1
nivelul 2: 1/2 2/1

- pe fiecare nivel k se plasează, sub fiecare fracție i/j de pe nivelul anterior, fracția $i / (i + j)$ în stânga și fracția $(i + j) / j$ în dreapta.

nivelul 1: 1/1
nivelul 2: 1/2 2/1
nivelul 3: 1/3 3/2 2/3 3/1

Cerință

Dându-se o fracție oarecare prin numărătorul și numitorul său, determinați numărul nivelului pe care se află fracția sau o fracție echivalentă (având aceeași valoare) cu aceasta.

Date de intrare

Fișier de intrare: **FRACTII.IN**
Linia 1: `N M`

- două numere naturale nenule, separate printr-un spațiu, reprezentând numărătorul și numitorul unei fracții (numărător respectiv numitor).

Date de ieșire

Fișier de ieșire: **FRACTII.OUT**
Linia 1: `niv`

- număr natural nenul, reprezentând numărul nivelului care corespunde fracției.

Restricție

- `1 < N,M ≤ 2000000000` (două miliarde)

Exemple

FRACTII.IN	FRACTII.OUT
13 8	6

FRACTII.IN	FRACTII.OUT
12 8	3

Timp de execuție: 1 secundă/test

P050103: Tablou

prof. Rodica Pintea, București

Generați un tablou bidimensional cu proprietățile:

- conține N linii și N coloane;
- elementele sale sunt numere naturale nenule;
- suma elementelor este egală cu numărul natural nenul S;
- pe nici o linie și pe nici o coloană nu există două elemente identice;
- diferența dintre cel mai mare și cel mai mic element ale tabloului este minimă.

Date de intrare

Fișier de intrare: **TABLOU.IN**
Linia 1: `N S`

- două numere naturale nenule, separate printr-un spațiu, reprezentând numărul de linii și de coloane ale tabloului, respectiv valoarea sumei tuturor elementelor din tablou.

Date de ieșire

Fișier de ieșire: **TABLOU.OUT**
Liniile 1...N: nr_{11} nr_{12} ... nr_{1N}
nr_{21} nr_{22} ... nr_{2N}
...
nr_{N1} nr_{N2} ... nr_{NN}

- pe aceste N linii se vor scrie elementele tabloului, câte o linie din tablou pe o linie din fișier; elementele se vor separa prin câte un spațiu.

Restricții

- `1 < N ≤ 100`
- `0 < S ≤` 2^{31}
- Dacă problema nu are soluție, în fișierul de ieșire se va scrie cifra `0`.
- Dacă problema are mai multe soluții, în fișier se va scrie una singură.

Exemplu

TABLOU.IN	TABLOU.OUT
3 51	4 6 7
	7 4 5
	5 7 6

Timp de execuție: 1 secundă/test

P050104: Competiție dificilă

Angel Proorocu, București

La o competiție au participat N concurenți. Fiecare dintre ei a primit un număr de concurs astfel încât să nu existe concurenți cu același număr. Numerele de concurs aparțin mulțimii $\{1, 2, ..., N\}$. Din păcate, clasamentul final a fost pierdut, iar comisia își poate aduce aminte doar câteva relații între unii participanți (de genul "participantul cu numărul 3 a ieșit înaintea celui cu numărul 5").

Cerință

Șeful comisiei are nevoie de un clasament final și vă cere să-l ajutați determinând primul clasament în ordine lexicografică ce respectă relațiile pe care și le amintește comisia.

Date de intrare

Fișier de intrare: **COMPET.IN**
Linia 1: `N M`

- două numere naturale nenule, reprezentând numărul concurenților, respectiv numărul relațiilor pe care și le amintește comisia;

Liniile 2...M+1: `i j`

- pe fiecare din aceste M linii se află câte două numere naturale nenule i și j, având semnificația: concurentul cu numărul de concurs i a fost în clasament înaintea concurentului cu numărul de concurs j.

Date de ieșire

Fișier de ieșire: **COMPET.OUT**

Linia 1: $nr_1\ nr_2\ \ldots\ nr_N$
- pe această linie se va scrie clasamentul sub forma unui șir de numere naturale nenule, separate prin câte un spațiu, reprezentând numerele de concurs ale concurenților, în ordine, de la primul clasat la ultimul.

Restricții
- 1< N ≤ 1000
- se garantează corectitudinea datelor de intrare și faptul că există totdeauna o soluție.

Exemple

COMPET.IN	COMPET.OUT
3 1 1 2	1 2 3

COMPET.IN	COMPET.OUT
4 2 2 1 3 4	2 1 3 4

Timp de execuție: 1 secundă/test

P050105: Cuvinte

prof. Maria Nițâ și prof. Adrian Nițâ, Oradea

Se consideră o listă având un număr cunoscut de cuvinte. Din această listă s-au ales două cuvinte oarecare. Se dorește transformarea primului cuvânt în cel de-al doilea, trecând prin cuvinte intermediare, existente în lista dată. Pentru transformarea unui cuvânt în altul se pot folosi următoarele operații: schimbarea, adăugarea sau ștergerea unui singur caracter.

Cerință
Dându-se o listă de cuvinte și două cuvinte din aceasta, găsiți cea mai scurtă secvență de operații care transformă primul cuvânt în cel de-al doilea folosind operațiile permise.

Date de intrare
Fișierul de intrare: CUVINTE.IN

Linia 1: N P Q
- trei numere naturale nenule, reprezentând numărul cuvintelor din listă (N), poziția primului cuvânt în listă (P), respectiv poziția celui de-al doilea cuvânt în listă (Q);

Liniile 2...N+1: cuvant
- aceste N linii conțin fiecare câte un cuvânt, aparținând listei.

Date de ieșire
Fișier de ieșire: CUVINTE.OUT

Linia 1: M
- număr natural, reprezentând numărul minim de pași necesari pentru a ajunge de la primul cuvânt la al doilea;

Liniile 2...M+2: cuv_i
- pe aceste linii apar în ordine cuvintele dintr-o secvență ce respectă cerința problemei (câte un cuvânt pe linie).

Restricții
- 2 ≤ N ≤ 100
- 1 ≤ M,P,Q ≤ 100
- numărul maxim de caractere dintr-un cuvânt este 100;
- dacă nu există soluție, în fișierul de ieșire se va scrie numărul 0 (zero);
- dacă sunt mai multe secvențe de lungime minimă, în fișier se va scrie una singură.

Exemple

CUVINTE.IN	CUVINTE.OUT
7 1 5 car cer cerc mar mare rosu inrosit	2 car mar mare

CUVINTE.IN	CUVINTE.OUT
7 1 6 car cer cerc mar mare	0

Timp de execuție: 2 secunde/test

P050106: Grup

prof. Emanuela Cerchez și prof. Marinel Șerban, Iași

Administratorul rețelei cu N calculatoare de la SRI împarte, din motive strategice, aceste calculatoare în mai multe grupuri. De fapt, important este doar numărul de grupuri și numărul de calculatoare din fiecare grup, așa că împărțirea este descrisă prin șirul numerelor de calculatoare din fiecare grup, ordonat crescător. Periodic, el procedează la o nouă împărțire pe grupe a calculatoarelor. Dintre toate împărțirile posibile ale calculatoarelor în grupuri putem alege ca următoare împărțire doar aceea a cărei descriere precede sau succede lexicografic imediat împărțirii curente.

Notă: Spunem că șirul $x_1x_2\ldots x_p$ precede lexicografic șirul $y_1y_2\ldots y_k$ dacă:

a) există un indice j, astfel încât $x_i = y_i$ pentru $i < j$ și $x_j < y_j$

sau

b) $p < k$ și $x_i = y_i$ pentru toți indicii $i \le p$

Exemple:

a) 3 7 2 5 precede lexicografic 3 7 4 1 6 2

b) 1 2 3 precede lexicografic 1 2 3 4

Cerință
Dându-se o împărțire în grupe a celor N calculatoare, determinați cele două variante candidate pentru împărțirea următoare.

Date de intrare

Fișier de intrare: **GRUP.IN**

Linia 1: `N k`

- numere naturale nenule, reprezentând numărul total (N) al calculatoarelor din rețea și numărul (k) de grupe.

Linia 2: $g_1\ g_2\ \ldots\ g_k$

- numărul calculatoarelor din fiecare grupă.

Date de ieșire

Fișier de ieșire: **GRUP.OUT**

Linia 1: `p`

- numărul de grupe din împărțirea care precede lexicografic imediat împărțirea dată;

Linia 2: $h_1\ h_2\ \ldots\ h_p$

- numărul de calculatoare din cele p grupe ale împărțirii precedente;

Linia 3: `u`

- numărul de grupe din împărțirea care succede lexicografic imediat împărțirea dată;

Linia 4: $t_1\ t_2\ \ldots\ t_u$

- numărul de calculatoare din cele u grupe ale împărțirii următoare.

Restricții

- $2 \le N \le 1000$
- $g_1+g_2+\ldots+g_k=h_1+h_2+\ldots+h_p=t_1+t_2+\ldots+t_u=N$
- $1 \le g_1 \le g_2 \le \ldots \le g_k;\ 1 \le h_1 \le h_2 \le \ldots \le h_p;\ 1 \le t_1 \le t_2 \le \ldots \le t_u$
- $1 < k < N$
- $1 \le p,u \le N$

Exemplu

GRUP.IN	GRUP.OUT
14 3	3
2 6 6	2 5 7
	2
	2 12

Timp de execuție: 1 secundă/test

Clasa a X-a

P050107: Alpinistul

Bogdan Stroe, București

Un alpinist ar vrea să cucerească un vârf cât mai înalt din Munții Carpați. Din păcate îi este frică să urce de pe o stâncă pe alta alăturată, dacă diferența de altitudine depășește 100 de metri. În schimb el coboară fără frică de pe o stâncă pe alta alăturată, indiferent de diferența de altitudine.

Alpinistul are harta muntelui pe care vrea să-l escaladeze. Harta este codificată sub forma unui caroiaj în care în pătrățele sunt notate altitudinile punctelor (înălțimile stâncilor) de pe hartă, date în metri față de nivelul mării. Alpinistul poate porni din orice pătrățel de pe hartă cu altitudine 0 (aflat la nivelul mării) și poate efectua un pas doar într-o porțiune de teren corespunzătoare unui pătrățel de pe hartă, învecinat pe verticală sau pe orizontală cu pătrățelul în care se află, cu condiția să nu îi fie frică.

Cerință

Alpinistul apelează la ajutorul vostru pentru a afla traseul de lungime minimă pe care trebuie să-l urmeze pentru a escalada un vârf cât mai înalt.

Date de intrare

Fișier de intrare: **ALPINIST.IN**

Linia 1: `M N`

- două numere naturale nenule, separate printr-un spațiu, reprezentând dimensiunile caroiajului corespunzător hărții;

Liniile 2...M+1: $v_1\ v_2\ \ldots\ v_N$

- pe aceste M linii sunt scrise câte N numere naturale, separate prin câte un spațiu, reprezentând valorile asociate caroiajului care codifică harta (linie după linie).

Date de ieșire

Fișier de ieșire: **ALPINIST.OUT**

Linia 1: `I`

- număr natural ce reprezintă înălțimea maximă la care poate ajunge alpinistul;

Linia 2: $X_P\ Y_P\ X_D\ Y_D$

- patru numere naturale nenule, separate prin câte un spațiu, reprezentând coordonatele pătrățelului din care pleacă alpinistul și coordonatele pătrățelului având înălțimea maximă în care poate ajunge; prin coordonatele pătrățelului se înțeleg numărul liniei și cel al coloanei pe care se află pătrățelul în caroiaj;

Linia 3: `L`

- număr natural reprezentând lungimea drumului; lungimea unui drum se definește ca fiind egal cu numărul pătrățelelor străbătute de alpinist;

Linia 4: $D_1\ D_2\ \ldots\ D_L$

- L caractere, separate prin câte un spațiu, reprezentând direcția în care se mișcă alpinistul la fiecare pas i, $i = 1, 2, \ldots, L$; caracterele pot fi: `'N'` (corespunzător unei mișcări în sus) `'S'` (corespunzător unei mișcări în jos), `'W'` (corespunzător unei mișcări la stânga) și `'E'` (corespunzător unei mișcări la dreapta).

Restricții

- $2 \le M,N \le 100$
- $0 \le v_i \le 1000$, $i = 1, 2, \ldots, N$ (pe fiecare din cele M linii)
- dacă sunt mai multe drumuri de lungime minimă, în fișier se va scrie unul singur.

Exemplu

ALPINIST.IN	ALPINIST.OUT
3 5	250
0 101 2 14 100	1 1 2 4
50 149 149 250 200	8
0 100 10 400 10	S E E N E E S W

Timp de execuție: 1 secundă/test

P050108: Asfaltare

prof. Roxana Tîmplaru, Craiova

Un primar proaspăt ales vrea să dovedească electoratului său că votându-l, cetățenii au făcut o alegere bună. În acest scop el a decis să reasfalteze străzile dintre N edificii importante din oraș, numerotate de la 1 la N. Între oricare două dintre aceste edificii există o singură stradă cu două sensuri de circulație. Edificiul numerotat cu 1 este primăria.

Primarul cere consilierilor să stabilească toate traseele pe care reasfaltarea străzilor o poate urma printre cele N edificii, știind că are H străzi "preferate" pe care trebuie să le asfalteze în mod obligatoriu. Se știe că oricare două străzi preferate nu au capete comune. Traseele care se vor reasfalta trebuie să pornească de la primărie, să ajungă o singură dată la fiecare din celelalte N-1 edificii și să se întoarcă tot la primărie.

Cerință

Determinați numărul traseelor distincte, respectând cerințele de mai sus.

Date de intrare

Fișier de intrare: **ASFALT.IN**

Linia 1: `N H`

- două numere naturale, separate printr-un spațiu, reprezentând numărul edificiilor (N), respectiv numărul străzilor preferate ale primarului (H).

Date de ieșire

Fișier de ieșire: **ASFALT.OUT**

Linia 1: `x`

- număr întreg pozitiv, reprezentând numărul traseelor distincte, posibile.

Restricții

- `3 ≤ N ≤ 1000`
- `0 ≤ H ≤ N/2`
- dacă un traseu este diferit de un altul doar prin direcția în care se parcurge drumul, pornind de la primărie și revenind aici, acesta se consideră identic cu primul. De exemplu, traseul `1-2-3-4-1` este identic cu traseul `1-4-3-2-1`.

Exemplu

ASFALT.IN	ASFALT.OUT
4 1	2

Timp de execuție: 5 secunde/test.

P050109: Oracolul decide!

prof. Doru Popescu Anastasiu, Slatina

La un concurs participă N concurenți. Fiecare concurent primește o foaie de hârtie pe care va scrie un cuvânt având cel mult 100 de caractere (litere mici ale alfabetului englez). Cuvintele vor fi distincte.

Pentru departajare, concurenții apelează la un oracol. Acesta produce și el un cuvânt. Va câștiga concurentul care a scris cuvântul "cel mai apropiat" de cel al oracolului.

Gradul de "apropiere" dintre două cuvinte este lungimea subcuvântului comun de lungime maximă. Prin subcuvânt al unui cuvânt dat se înțelege un cuvânt care se poate obține din cuvântul dat, eliminând 0 sau mai multe litere și păstrând ordinea literelor rămase.

Cerință

Se cunosc cuvântul c_0 produs de oracol și cuvintele c_i, i = 1, ..., N scrise de concurenți. Pentru a ajuta comisia să desemneze câștigătorul, se cere ca pentru fiecare valoare i să identificați pozițiile literelor ce trebuie șterse din c_0 și din c_i astfel încât prin ștergere să se obțină unul dintre subcuvintele comune de lungime maximă.

Date de intrare

Fișier de intrare: **ORACOL.IN**

Linia 1: `N`

- număr natural nenul, reprezentând numărul concurenților;

Linia 2: c_0

- cuvântul produs de oracol;

Liniile 3...N+2: `cuvant`

- pe aceste N linii se află cuvintele scrise de cei N concurenți, un cuvânt pe o linie.

Date de ieșire

Fișier de ieșire: **ORACOL.OUT**

Liniile 1...2*N: *pozițiile literelor ce trebuie șterse*

- pe fiecare linie i (i = 1, 3, ..., 2*N - 1) se vor scrie numere naturale nenule, separate prin câte un spațiu, reprezentând pozițiile de pe care se vor șterge litere din cuvântul produs de oracol; pe fiecare linie j (j = 2, 4, ..., 2*N) se vor scrie numere naturale nenule, separate prin câte un spațiu, reprezentând pozițiile de pe care se vor șterge litere din cuvântul concurentului cu numărul j/2.

Restricții

- `2 ≤ N ≤ 100`
- dacă există mai multe soluții, în fișier se va scrie una singură.
- dacă dintr-un cuvânt nu se va tăia nici o literă, linia respectivă din fișierul de intrare va rămâne vidă.

Exemplu

ORACOL.IN	ORACOL.OUT
3	3
abc	3 4
abxd	
aabxyc	1 4 5
acb	3
	2

Timp de execuție: 1 secundă/test

P050110: Alipiri

prof. Gelu Manolache, Piatra Neamț

Spunem că numărul natural nenul n_1 se poate alipi pe K biți cu numărul natural nenul n_2 dacă reprezentările lor binare au fiecare cel puțin K biți (primul bit fiind 1), iar ultimii K biți ai numărului n_1 coincid cu primii K biți ai numărului n_2. După alipire rezultă un alt număr, format din concatenarea lui n_1 cu n_2, dar în care secvența comună de K biți apare o singură dată și în ea fiecare bit este înlocuit cu complementarul său (0 devine 1 și invers).

Exemplu: Numărul $78=1001110_2$ se poate alipi pe trei biți cu $25=11001_2$ și rezultă numărul $100100101_2=293$.

Numerele $nr_1, nr_2, ..., nr_M$ formează o listă de alipiri pe K biți, de lungimea $M \geq 1$, cu finalul P, dacă nr_1 se alipește pe K biți cu nr_2, rezultatul se alipește pe K biți cu nr_3, ..., rezultatul se alipește pe K biți cu nr_M, iar rezultatul final este numărul P.

Notă: Dacă $M = 1$, atunci trebuie să avem $nr_1 = P$.

Cerințe

Pentru un șir dat de numere naturale nenule și valorile naturale nenule K și P, se cere:

a) să se determine lungimea maximă a unei liste de alipiri pe K biți, formată cu elemente din șirul dat, nu neapărat în ordinea din acest șir;

b) să se verifice dacă se poate obține, cu numere din șirul dat, o listă de alipiri pe K biți cu finalul P; în caz afirmativ să se precizeze o listă de lungime minimă de alipiri pe K biți cu finalul P.

Date de intrare

Fișier de intrare: **ALIPIRI.IN**

Linia 1: `N K P`

- trei numere naturale nenule, separate prin câte un spațiu, reprezentând numărul N al numerelor date, lungimea K a secvențelor comune din alipiri, respectiv valoarea P a numărului care trebuie obținut prin alipiri;

Linia 2: nr_1 nr_2 ... nr_N

- N numere naturale nenule, separate prin câte un spațiu, reprezentând șirul dat.

Date de ieșire

Fișier de ieșire: **ALIPIRI.OUT**

Linia 1: `M`

- număr natural nenul, reprezentând rezultatul de la cerința a);

Linia 2: *mesaj*

- dacă cerința b) poate fi satisfăcută, pe această linie se va scrie `DA`, în caz contrar se va scrie `NU`;

Linia 3: nr_1 nr_2 ...

- dacă pe linia 2 s-a scris `DA`, pe această linie se va scrie o listă de alipiri (pe K biți) de lungime minimă cu finalul P; numerele din listă se vor separa prin câte un spațiu.

Restricții

- $1 \leq K \leq 8, 1 \leq N \leq 9, 1 \leq P \leq 2000000000$
- dacă cerința b) este satisfăcută de mai multe liste (de lungime minimă), în fișier se va scrie una singură;
- pentru ambele cerințe fiecare număr din șirul de intrare poate apărea cel mult o dată în listele de alipiri considerate.

Exemplu

ALIPIRI.IN	ALIPIRI.OUT
4 3 291	4
14 59 36 31	DA
	59 31 14

Timp de execuție: 10 secunde/test

P050111: Drum

prof. Doru Popescu Anastasiu, Slatina

În regatul *XLand* orașele erau înconjurate de ziduri în formă de poligoane convexe. Împăratul a dispus construirea unui drum de legătură directă între capitală și un alt oraș dat. Fiecare extremitate a drumului poate fi orice punct situat pe zidul orașului, respectiv capitalei. Lungimea drumului este distanța dintre extremitățile sale.

Cerință

Determinați cel mai scurt drum dintre capitală și orașul dat.

Date de intrare

Fișier de intrare: **DRUM.IN**

Linia 1: `K1`

- număr natural nenul, reprezentând numărul de colțuri ale zidurilor capitalei;

Linia 2: x_1 y_1 x_2 y_2 ... x_{K1} y_{K1}

- $K1$ perechi de numere întregi, separate prin câte un spațiu, reprezentând coordonatele vârfurilor zidurilor capitalei;

Linia 3: `K2`

- număr natural nenul, reprezentând numărul de colțuri ale zidurilor orașului dat;

Linia 4: x_1 y_1 x_2 y_2 ... x_{K2} y_{K2}

- $K2$ perechi de numere întregi, separate prin câte un spațiu, reprezentând coordonatele vârfurilor zidurilor acestui oraș.

Date de ieșire

Fișier de ieșire: `DRUM.OUT`

Linia 1: x_1 y_1 x_2 y_2

- patru numere reale trunchiate la patru zecimale, separate prin câte un spațiu, reprezentând extremitățile drumului de legătură respectiv.

Restricții

- $2 \leq K1, K2 \leq 20$
- coordonatele vârfurilor zidurilor ce înconjoară orașul, respectiv capitala sunt numere întregi aparținând intervalului [-100, 100] și sunt date fie în sensul deplasării

acelor de ceasornic, fie în sens invers deplasării acelor de ceasornic;
- capitala și orașul nu au nici un punct comun (nu au puncte interioare comune și nu au puncte comune pe zidurile lor).

Exemplu

DRUM.IN	DRUM.OUT
4 3 4 3 2 5 2 5 4 4 8 3 8 6 11 6 11 3	5.0000 3.5000 8.0000 3.5000

Timp de execuție: 1 secundă/test

P050112: Pavări

prof. Doru Popescu Anastasiu, Slatina

Se dă un dreptunghi cu lungimea egală cu $2N$ centimetri și lățimea egală cu trei centimetri.

Cerință

Să se determine numărul M al pavărilor distincte cu dale dreptunghiulare care au lungimea egală cu un centimetru și lățimea egală cu doi centimetri.

Date de intrare

Fișier de intrare: **PAVARI.IN**

Linia 1: N
- număr natural nenul, reprezentând jumătatea lungimii dreptunghiului.

Date de ieșire

Fișier de ieșire: **PAVARI.OUT**

Linia 1: M
- număr natural nenul, reprezentând numărul modalităților de a pava dreptunghiul.

Restricții

- $1 \le N \le 100$

Exemplu

PAVARI.IN	PAVARI.OUT
2	11

Timp de execuție: 1 secundă/test

Clasele a XI-a și a XII-a

P050113: Comparări

Bogdan Dumitru, București

Fie un șir de N numere întregi. Numim al doilea maxim al șirului cel mai mare număr din șir, cu excepția maximului. Pentru a afla maximul și al doilea maxim, veți efectua comparări între elementele șirului.

Având la dispoziție numai $N + \lceil \log_2 N \rceil - 2$ comparări, aflați pozițiile pe care le ocupă maximul și al doilea maxim. Prin $\lceil \log_2 N \rceil$ s-a notat *partea întreagă superioară a lui "logaritm în baza 2 din N"*.

Cerință

Aflați cele două numere, efectuând cel mult $N + \lceil \log_2 N \rceil - 2$ comparări. Pentru a putea efectua comparările, comisia vă pune la dispoziție funcția `int Compara(int i,int j)` (pentru cei care programează în *C* sau *C++*), respectiv funcția `Compara(i,j:Integer):Integer;` (pentru programatorii în *Pascal*), care compară numărul de pe poziția *i* cu cel de pe poziția *j* și returnează 1, dacă elementul aflat pe poziția *i* este mai mare sau egal cu cel de pe poziția *j*, respectiv -1, în caz contrar.

Programul trebuie să apeleze la început funcția `GetN` care returnează numărul de elemente din șir. Apelul va fi de forma `N=GetN()`, pentru programatorii în *C/C++*, respectiv `N:=GetN`, pentru programatorii în *Pascal*, unde am presupus că *N* este variabilă de tip `int`, respectiv `Integer`, în care este memorată lungimea șirului. Este esențial ca apelul funcției `GetN` să preceadă apelurile funcției `Compara` și, în plus, să existe un singur apel al funcției `GetN`.

Programatorii în *C/C++* vor găsi fișierul **COMPAR.H**, iar cei care lucrează în *Pascal* vor găsi fișierul **COMPAR.PAS** reprezentând sursele unit-ului comisiei.

În fișierul **COMPAR.OUT** veți scrie pozițiile pe care se află maximul șirului și al doilea maxim, separate printr-un spațiu.

Pentru a folosi unit-ul, trebuie să creați fișierul **COMPAR.IN**, în care veți scrie pe prima linie numărul *N*, iar pe a doua linie elementele șirului, separate prin câte un spațiu.

Versiunea *Pascal* a unit-ului este:

```
Unit compar;

interface

const my_nmax=1000;
var my_a:array[1..my_nmax+20] of Integer;
    my_apelat,my_n,my_napel,my_apel:Integer;
function GetN:Integer;
function compara(i,j:Integer):Integer;

implementation

function GetN:Integer;
var f:Text;
    i,log,tmp:Integer;
begin
  my_apelat:=1;
  Assign(f,'COMPAR.IN');
  Reset(f);
  Readln(f,my_n);
  for i:=1 to my_n do
    Read(f,my_a[i]);
```

```
  Close(f);
  my_apel:=0;
  tmp:=1;
  log:=0;
  while tmp<my_n do
  begin
    tmp:=tmp*2;
    log:=log+1
  end;
  my_napel:=my_n+log-2;
  getn:=my_n
end;

function compara(i,j:Integer):Integer;
var f:Text;
begin
  if my_apelat=0 then
  begin
    Writeln('Nu ati apelat functia GetN!');
    Halt
  end;
  my_apel:=my_apel+1;
  if my_apel>my_napel then
  begin
    Writeln('Prea multe apeluri!');
    Halt
  end;
  if (i<1) or (i>my_n) or
     (j<1) or (j>my_n) then
  begin
    Writeln('Apel ilegal!');
    Halt
  end;
  if my_a[i]>=my_a[j] then
    compara:=1
  else compara:=-1
end;

End.
```

Implementarea *C/C++* a aceleiași funcții este următoarea:

```
#include <stdio.h>
#include <math.h>
#include <process.h>
#define my_nmax 1000

int my_a[my_nmax+20];
int my_apelat,my_n,my_napel,my_apel;

extern int GetN(void){
  FILE *f;
  int i,log,tmp;
  my_apelat=1;
  f=fopen("COMPAR.IN","r");
  fscanf(f,"%d",&my_n);
  for (i=1;i<=my_n;i++)
    fscanf(f,"%d",&my_a[i]);
  fclose(f);
  my_apel=0;
  tmp=1;
  log=0;
  while (tmp<my_n){
    tmp=tmp*2;
    log++;
  }
  my_napel=my_n+log-2;
  return my_n;
}

extern int Compara(int i,int j){
  FILE *f;
  if (!my_apelat){
    puts("Nu ati apelat functia GetN()!");
    exit(0);
  }
  my_apel=my_apel+1;
  if (my_apel>my_napel){
    puts("Prea multe apeluri!");
    exit(0);
  }
  if (i<1 || i>my_n || j<1 || j>my_n){
    puts("Apel ilegal!");
    exit(0);
  }
  if (my_a[i]>=my_a[j])
    return 1;
  else return -1;
}
```

Restricții

- $2 \leq N \leq 1000$

Exemple

Dacă șirul considerat este `5 3 4`, atunci o executare corectă a programului este următoarea:

`N = GetN()`, obțineți `N=3`;
`Compara(1,2)`, se returnează `1`;
`Compara(1,3)`, se returnează `1`;
`Compara(2,3)`, se returnează `-1`;

În acest moment ați epuizat numărul de comparări permise (3+2-2=3 comparări) și în fișierul **COMPAR.OUT** veți scrie numerele `1` și `3`, separate printr-un spațiu, reprezentând pozițiile pe care se află maximul șirului (5), respectiv al doilea maxim (4).

Dacă șirul considerat este (5, 7, 7), o executare corectă a programului este următoarea:
`N = GetN()`, obțineți `N=3`;
`Compara(1,2)`, se returnează `-1`;

`Compara(2,3)`, se returnează 1;
`Compara(1,3)`, se returnează -1;

În acest moment ați epuizat numărul de comparări permise (3+2-2=3 comparări) și în fișierul **COMPAR.OUT** veți scrie numerele 2 și 3, separate printr-un spațiu, reprezentând pozițiile pe care se află maximul șirului (7), respectiv al doilea maxim (7).

Observație

Pentru acest exemplu, mai există o soluție corectă, cea în care considerați maximul (7) pe poziția 3 și al doilea maxim (7) pe poziția 2.

Timp de execuție: 1 secundă/test

Se garantează că `1008` apeluri ale funcției `Compara`, împreună cu un apel al funcției `GetN` nu durează mai mult de 0.1 secunde.

P050114: Relee

șef lucrări Stelian Ciurea, Sibiu

Fie date altitudinile a N puncte, situate în linie dreaptă de-a lungul axei Ox, astfel încât ele corespund unor abscise naturale consecutive. Primul punct are abscisa `1`. Din acest punct trebuie trimisă o rază laser în ultimul punct (cel de abscisă N). Raza se propagă doar în linie dreaptă. Pentru a "ocoli" punctele având altitudini care împiedică trecerea razei, în anumite puncte se montează relee care schimbă unghiul sub care se propagă raza, cu scopul ca ea să poată trece de vârfurile care se află în drumul ei.

Releele se vor monta în oricare dintre punctele date, mai puțin în primul punct, de unde raza poate porni sub orice unghi și în ultimul punct unde raza poate fi recepționată sub orice unghi.

S-a observat că dacă în anumite puncte releul se montează pe un pilon, numărul releelor necesare se poate micșora. Toți pilonii care se vor monta au aceeași înălțime dată H.

Cerință

Determinați numărul minim de relee pentru ca raza să ajungă din punctul inițial în cel final, precum și punctele în care acestea se vor monta. În cazul în care există mai multe soluții cu același număr minim de relee, se va alege cea cu număr minim de piloni.

Date de intrare

Fișier de intrare: **RELEE.IN**

Linia 1: `N H`
- două numere naturale nenule, reprezentând numărul punctelor (`N`), respectiv înălțimea pilonilor (`H`);

Linia 2: $A_1\ A_2\ \ldots\ A_N$
- N numere întregi, separate prin câte un spațiu, reprezentând altitudinile (înălțimile) punctelor.

Date de ieșire

Fișier de ieșire: **RELEE.OUT**

Linia 1: `NRrelee`
- număr natural, reprezentând numărul releelor care se vor monta, fără să fie înălțate pe piloni;

Linia 2: `NRpiloni`
- număr natural, reprezentând numărul pilonilor care se vor monta;

Linia 3: $C_1\ C_2\ \ldots\ C_{NRrelee}$
- numere naturale nenule, reprezentând numărul de ordine al punctelor unde se vor monta relee, fără să fie înălțate pe piloni;

Linia 4: $D_1\ D_2\ \ldots\ D_{NRpiloni}$
- numere naturale nenule, reprezentând numărul de ordine al punctelor unde releele se vor monta pe piloni.

Dacă valoarea `NRrelee` este `0` atunci linia a treia va fi vidă, iar dacă valoarea `NRpiloni` este `0`, atunci linia a patra va fi vidă.

Restricții

- $1 \le N \le 200$
- $1 \le H \le 500$
- $1 \le A_i \le 2500$ $i = 1, 2, \ldots, N$
- dacă trei vârfuri sunt coliniare, atunci pe cel din mijloc nu trebuie amplasat un releu.

Exemplu

RELEE.IN	RELEE.OUT
9 2	1
3 2 6 6 4 3 5 3 2	1
	7
	4

Timp de execuție: 2 secunde/test

P050115: Telecomanda

prof. Emanuela Cerchez și prof. Marinel Șerban, Iași

Cu ocazia olimpiadei, televiziunea locală organizează un nou joc în direct. Organizatorii utilizează un calculator, care generează și afișează pe un monitor două numere de cel mult 100 de cifre fiecare ($N1$ și $N2$). Fiecare concurent dispune de o telecomandă prevăzută cu un afișaj de o cifră și cu anumite taste. Telecomanda are și o memorie, în care sunt reținute în ordine cifrele obținute de concurenți. Cifrele primului număr ($N1$) sunt afișate succesiv pe afișajul telecomenzii fiecărui concurent, în ordine de la stânga la dreapta. Concurenții trebuie să transforme primul număr, obținând în memoria telecomenzii proprii pe cel de al doilea, utilizând tastele pe care le au la dispoziție pe telecomandă. După efectuarea unei operații asupra cifrei curente (cea de pe afișaj), pe afișaj apare automat următoarea cifră din $N1$ (dacă mai există).Efectele apăsării tastelor sunt:
- + *urmat de o cifră*: se generează suma dintre cifra de pe afișaj și cifra tastată (operație posibilă doar dacă suma este tot o cifră); cifra sumă este reținută în memorie;
- - *urmat de o cifră*: se generează diferența dintre cifra de pe afișaj și cifra tastată (operație posibilă doar dacă se obține tot o cifră); cifra obținută este reținută în memorie;

- * *urmat de o cifră*: se reține în memorie valoarea tastei care se acționează după tasta *; deoarece asupra cifrei curente din $N1$ nu se efectuează nici o operație, aceasta nu dispare de pe afișaj;
- /: șterge cifra curentă din $N1$;
- #: se șterg din $N1$ cifra curentă și toate cifrele care urmează, până la sfârșit.
- =: se copiază în memorie cifra curentă.

Acțiunea se încheie atunci când toate cifrele lui $N1$ au fost prelucrate. Se obține o soluție dacă în memoria telecomenzii se află cifrele numărului $N2$. O soluție este optimă dacă numărul de taste acționate este minim. Câștigătorii jocului sunt acei concurenți care descoperă o soluție optimă.

Cerință

Date fiind $N1$ și $N2$, scrieți un program care să determine o soluție optimă de transformare a numărului $N1$ în numărul $N2$.

Date de intrare

Fișier de intrare: **TELE.IN**

Linia 1: `N1`

- numărul inițial

Linia 2: `N2`

Date de ieșire

Fișier de ieșire **TELE.OUT**

Linia 1: `min`

- număr natural nenul, reprezentând numărul minim de taste acționate pentru transformarea lui $N1$ în $N2$.

Linia 2: $t_1 t_2 \ldots t_{min}$

- succesiune de `min` caractere, care reprezintă tastele acționate; între caractere nu vor apărea separatori.

Exemplu

TELE.IN	TELE.OUT
372	4
78	/=+6

Timp de execuție: 1 secundă/test

P050116: Entries

Dumitru Bogdan, București

Se consideră un graf care inițial este format din P noduri izolate, etichetate de la 1 la P. Se mai consideră N intrări, unde intrare poate însemna:

- *comandă* - o comandă are forma `I+J`, cu semnificația că în graf se adaugă muchia care unește nodurile I și J (dacă I și J erau deja unite în acel moment, nu se întreprinde nici o acțiune);
- *întrebare* - o întrebare este de forma `'I?J'`, adică se întreabă dacă în acel moment I și J sunt în aceeași componentă conexă.

Se pleacă deci de la un graf inițial format din noduri izolate, care pe parcurs se "unifică". Tot pe parcurs sunteți întrebați dacă anumite perechi de noduri sunt sau nu în aceeași componentă conexă.

Din fișierul **ENTRIES.IN** veți citi de pe prima linie numărul N de intrări. Pe următoarele N linii se găsesc intrările, câte una pe linie. O intrare este codificată prin trei numere separate prin câte un blanc. Primele două numere reprezintă nodurile I și J (numere întregi, cuprinse între 1 și P), iar al treilea este `1` dacă intrarea este o comandă, respectiv `2` dacă intrarea este o întrebare.

La fiecare întrebare, veți scrie pe o linie separată în fișierul **ENTRIES.OUT** numărul `1` dacă nodurile despre care ați fost întrebați sunt în acel moment în aceeași componentă conexă, respectiv numărul `0` în caz contrar.

Restricții

- `1 ≤ N ≤ 5000`
- `1 ≤ P ≤ 10000000`

Exemplu

ENTRIES.IN	ENTRIES.OUT
9	0
1 2 2	0
1 2 1	1
3 7 2	0
2 3 1	1
1 3 2	0
2 4 2	
1 4 1	
3 4 2	
1 7 2	

Timp de execuție: 1 secundă/test

P050117: Robot

prof. Emanuela Cerchez, Iași

Un robot punctiform se poate deplasa, în plan, în linie dreaptă în orice direcție. Robotul se găsește într-o poziție inițială S și trebuie să ajungă într-o poziție finală F, evitând coliziunile cu obstacolele existente în teren. Obstacolele sunt suprafețe poligonale convexe, cu interioarele și frontierele disjuncte. Spunem că robotul a intrat în coliziune cu un obstacol dacă poziția sa devine interioară obstacolului. Prin urmare, dacă robotul se deplasează de-a lungul unui obstacol, nu intră în coliziune cu acesta.

Cerință

Scrieți un program care să determine cel mai scurt traseu pe care robotul îl poate urma de la poziția sa inițială S la poziția sa finală F, fără a intra în coliziune cu nici un obstacol. Traseul va fi precizat prin succesiunea punctelor critice (punctul inițial, punctele în care robotul își schimbă direcția și punctul final). Lungimea traseului este egală cu suma lungimilor segmentelor care îl compun.

Date de intrare

Fișier de intrare: **ROBOT.IN**

Linia 1: x_S y_S
• coordonatele poziției inițiale a robotului;
Linia 2: x_F y_F
• coordonatele poziției finale a robotului;
Linia 3: n
• numărul de obstacole;
Linia 4: k_1
• numărul de vârfuri ale primului obstacol;
Liniile 5...4+k_1: x_{i1} y_{i1}
• k_1 linii care conțin câte două numere care reprezintă coordonatele vârfurilor primului obstacol;
Linia 5+k_1: k_2
• numărul de vârfuri ale celui de-al doilea obstacol;
Liniile 6+k_1...5+k_1+k_2: x_{i2} y_{i2}
• k_2 linii care conțin câte două numere care reprezintă coordonatele vârfurilor celui de-al doilea obstacol;
...
Linia 4+n+k_1+...+k_{n-1}: k_n
• numărul de vârfuri ale ultimului obstacol;
Liniile 5+n+k_1+...+k_{n-1}...4+n+k_1+...+k_n: x_{in} y_{in}
• k_n linii care conțin câte două numere care reprezintă coordonatele vârfurilor ultimului obstacol.

Date de ieșire

Fișier de ieșire: **ROBOT.OUT**
Linia 1: nr
• numărul de puncte de pe traseu;
Liniile 2..nr+1: x_i y_i
• coordonatele punctelor critice de pe traseu.

Restricții

• n este un număr natural, $0 \le N \le 50$
• $k_1+k_2+...+k_n \le 200$
• x_i, y_i sunt numere reale, $|x_i|, |y_i| \le 100000$
• punctele S și F nu se află în interiorul nici unui obstacol
• coordonatele se vor scrie în fișierul de ieșire cu trei zecimale semnificative
• dacă există mai multe trasee de lungime minimă, în fișierul de ieșire se va scrie o singură soluție.

Exemplu

ROBOT.IN
```
10 5
-10 -10
1
3
0 0
0 10
10.5 0
```

ROBOT.OUT
```
3
10 5
10.5 0
-10 -10
```

Timp de execuție: 1 secundă/test

P050118: Text mare

Radu Ștefan, Brașov

Se dă un șir de caractere, ce reprezintă o frază; între cuvinte nu apar separatori. De asemenea, se dă un vocabular având cel mult 32000 de cuvinte; fiecare cuvânt este format din cel mult 16 caractere. Fraza poate avea cel mult 32000 caractere, care sunt litere mici din alfabetul latin. Cuvintele din vocabular nu sunt neapărat diferite și nu apar într-o ordine prestabilită.

Cerință

Despărțiți fraza într-un număr minim de cuvinte; toate aceste cuvinte trebuie să existe în vocabularul dat.

Date de intrare

Fișier de intrare: **TEXTMARE.IN**
Linia 1: fraza
• șir de caractere terminat cu punct, reprezentând fraza
Liniile 2...: cuvant
• următoarele linii conțin câte un cuvânt din vocabular; fișierul se termină cu o linie liberă.

Date de ieșire

Fișier de ieșire: **TEXTMARE.OUT**
Linia 1: fraza
• șir de caractere terminat cu punct, reprezentând fraza despărțită în cuvinte; între oricare două cuvinte consecutive va apărea exact câte un blanc.

Restricții

• dacă există mai multe soluții, la ieșire va fi scrisă una singură;
• dacă nu există soluție, în fișierul de ieșire se va scrie doar cifra 0 (zero);
• într-o frază un cuvânt poate să apară de mai multe ori, fiecare apariție a sa fiind numărată.

Exemplu

TEXTMARE.IN
```
acestaesteuntext.
text
acesta
acest
a
care
este
un
```

TEXTMARE.OUT
```
acesta este un text.
```

Timp de execuție: 2 secunde/test

Barajul de selecție

P050119: Deșert

Radu Ștefan, Brașov

Se dă o imagine alb-negru care reprezintă fotografia, realizată din satelit, a unei zone dintr-un deșert.

În această zonă se caută o construcție secretă de formă dreptunghiulară. Imaginea porțiunii de deșert analizate a

fost codificată într-o matrice având dimensiunile maxime de $N \times 255$ elemente. Imaginea construcției este codificată într-o matrice având $K \times 32$ elemente. Elementele celor două matrice pot fi numai caracterele `'.'` și `'#'`.

Cerință

Determinați de câte ori se regăsește fotografia construcției pe hartă.

Date de intrare

Fișier de intrare: **DESERT.IN**

Linia 1: `N K`

- două numere naturale nenule, N reprezentând numărul de linii al matricei care codifică fotografia deșertului, iar K numărul de linii al matricei pătratice care codifică fotografia construcției;

Liniile 2...K+1: $c_{i,1}\ c_{i,2}\ \ldots\ c_{i,32}$

- aceste K linii conțin în formă codificată matricea fotografiei construcției (32 de caractere `'#'` și `'.'` neseparate prin spații);

Liniile K+2...K+N+1: $d_{i,1}\ d_{i,2}\ \ldots\ d_{i,255}$

- aceste N linii conțin în formă codificată matricea fotografiei deșertului: (255 de caractere `'#'` și `'.'` neseparate prin spații).

Date de ieșire

Fișier de ieșire: **DESERT.OUT**

Linia 1: `NR`

- număr natural, reprezentând numărul de potriviri ale matricei fotografiei construcției în matricea fotografiei deșertului.

Restricții

- $3 \le N \le 1024$
- $1 < K < N$
- Fotografia construcției nu se va roti și nu se va oglindi.

Exemplu

DESERT.IN	DESERT.OUT
3 2	2
#....(*în total* 31 *de puncte*)....	
#....(*în total* 31 *de puncte*)....	
#....(*în total* 254 *de puncte*)....	
#....(*în total* 254 *de puncte*)....	
#....(*în total* 254 *de puncte*)....	

Timp de execuție: 1 secundă/test

P050120: Potrivire

prof. Tudor Sorin, București

Se consideră două numere naturale nenule N și S.

Cerință

Determinați numerele distincte $x_1, x_2, ..., x_N$ aparținând mulțimii $\{1, 2, ..., N\}$ astfel încât

$$1 \cdot x_1 + 2 \cdot x_2 + ... + N \cdot x_N = S.$$

Date de intrare

Fișier de intrare: **POTRIV.IN**

Linia 1: `N S`

- două numere naturale nenule, separate printr-un spațiu, reprezentând numărul dat, respectiv suma dată.

Date de ieșire

Fișier de ieșire: **POTRIV.OUT**

Linia 1: $x_1\ x_2\ \ldots\ x_N$

- N numere naturale nenule, separate prin câte un spațiu, reprezentând soluția problemei. Dacă nu există soluție, pe această linie se va scrie numărul `0`.

Restricții

- $3 < N \le 1000$
- $0 < S \le 333834000$
- dacă există mai multe soluții, în fișier se va scrie una singură.

Exemplu

POTRIV.IN	POTRIV.OUT
4 26	3 2 1 4

Explicație

`1•3 + 2•2 + 3•1 + 4•4 = 26`

Timp de execuție: 1 secundă/test

P050121: Tort

Mihai Stroe, București

Gigel a primit de ziua lui un tort dreptunghiular pe care mama lui a desenat un caroiaj. În mijlocul anumitor pătrățele sunt așezate bezele, una într-un pătrățel. Gigel ar vrea să mănânce cât mai multe bezele, dar tatăl lui a inventat o șmecherie prin care speră să reducă pofta de bezele a lui Gigel.

Gigel a primit și o mulțime de bucăți de ciocolată. Tatăl lui îi cere să înlocuiască bezelele pe care vrea să le mănânce cu bucăți de ciocolată, formate din două pătrățele. Gigel trebuie să înlocuiască întotdeauna câte două bezele (vecine pe orizontală sau verticală), fără a suprapune bucățile de ciocolată.

Cerință

Știind că există suficiente bucăți de ciocolată, ajutați-l pe Gigel să înlocuiască cât mai multe bezele, respectând regulile impuse de tatăl lui.

Date de intrare

Fișier de intrare: **TORT.IN**

Linia 1: `M N`

- două numere naturale nenule, separate printr-un spațiu, reprezentând dimensiunile caroiajului de pe tort;

Liniile 2...M+1: $t_{i1}\ t_{i2}\ \ldots\ t_{iN}$

- aceste M linii conțin codificarea caroiajului de pe tort, linie după linie: valoarea 1 reprezintă o bezea, 0 înseam-

nă că în pătrățelul respectiv nu există bezea; între două valori numerice există un singur spațiu.

Date de ieșire
Fișier de ieșire: **TORT.OUT**
Linia 1: K
- număr natural, reprezentând numărul bucăților de ciocolată (o bucată este formată din două pătrățele, deci va înlocui două bezele); acest număr trebuie să fie cel mai mare posibil, respectând cerințele problemei; o bucată de ciocolată, formată din două pătrățele nu se poate rupe;

Liniile 2...*K*+1: X_i Y_i D_i
- aceste *K* linii conțin câte două numere naturale nenule și un caracter, separate prin câte un spațiu, care descriu amplasarea unei bucăți de ciocolată; X_i și Y_i sunt coordonatele pătrățelului stânga sus, iar D_i identifică poziția celuilalt pătrățel al bucății astfel: este 'E' (*Est*) sau 'S' (*Sud*), după cum al doilea pătrățel se află la dreapta primului sau sub primul.

Restricții
- 1 < M,N ≤ 100
- o bucată de ciocolată trebuie să înlocuiască exact două bezele
- bucățile de ciocolată nu se pot suprapune
- ordinea în fișierul de ieșire a descrierii bucăților de ciocolată poate fi oarecare.

Exemplu

TORT.IN	TORT.OUT
4 3	3
1 0 1	2 2 S
0 1 0	3 1 S
1 1 0	4 2 E
1 1 1	

Timp de execuție: 1 secundă/test

P050122: Descompuneri

șef lucrări Stelian Ciurea, Sibiu

Se dau numerele naturale nenule *N*, *K* și *X*. Numim o *K*-descompunere a lui *N* un șir strict crescător de *K* numere naturale nenule, a căror sumă este *N*.

Cerință
Dintre toate *K*-descompunerile lui *N*, să se determine în câte apare numărul *X*.

Date de intrare
Fișier de intrare: **DESCOMP.IN**
Linia 1: N K X
- trei numere naturale nenule, separate prin câte un spațiu, având semnificația din enunț.

Date de ieșire
Fișier de ieșire: **DESCOMP.OUT**
Linia 1: NR
- număr natural, reprezentând numărul descompunerilor respectând cerințele problemei.

Restricții
- 2 ≤ N ≤ 350

Exemplu

DESCOMP.IN	DESCOMP.OUT
12 3 2	3

Explicații
12=1+**2**+9
12=**2**+3+7
12=**2**+4+6

Timp de execuție: 1 secundă/test

P050123: James Bond

Mihai Stroe, București

Un grup de teroriști s-a ascuns într-un sistem de canalizare. James Bond, având la dispoziție o hartă care descrie configurația sistemului de canalizare și pozițiile în care teroriștii au amplasat bombe, vrea să anihileze grupul de teroriști. Conductele din acest sistem sunt desenate pe hartă sub forma unor segmente de dreaptă în plan, astfel încât oricare două segmente au cel mult un punct comun, care este capăt pentru amândouă segmentele.

Explozia unei bombe afectează o zonă circulară (interior și frontieră) și este instantanee pe întreaga zonă afectată. Bombele pot să difere una de cealaltă prin raza de acțiune sau prin timpul la care explodează.

În urma unei explozii, canalele care trec prin zona de acțiune a bombei devin impracticabile pe segmentul afectat, dar, dacă Bond se află într-un astfel de canal și a trecut deja de zona afectată de explozie, poate să-și continue drumul.

Dacă James Bond se află în raza de acțiune a unei bombe la momentul exploziei, el moare.

La momentul inițial (momentul 0), James Bond pornește dintr-un punct al sistemului de canalizare (capăt al unui segment) și se deplasează cu viteză constantă, egală cu 1. James Bond trebuie să ajungă în punctul în care se află teroriștii (capăt al unui segment) în cel mai scurt timp posibil.

Cerință
Determinați calea pe care trebuie să o urmeze James Bond astfel încât să ajungă viu la teroriști în cel mai scurt timp.

Date de intrare
Fișier de intrare: **BOND.IN**
Linia 1: M N
- două numere întregi, pozitive, separate printr-un spațiu, reprezentând numărul de conducte și numărul de bombe;

Liniile 2...M+1: x_1 y_1 x_2 y_2
- aceste M linii conțin fiecare câte patru numere întregi, separate prin câte un spațiu, reprezentând descrierea segmentelor prin coordonatele extremităților;

Liniile M+2...M+N+1: x_c y_c R T
- aceste N linii conțin fiecare câte patru numere întregi, separate prin câte un spațiu, reprezentând descrierea bombelor prin coordonatele punctului unde sunt amplasate, raza cercului pe care acționează și momentul de timp la care acționează;

Linia M+N+2: x_p y_p
- două numere întregi, separate printr-un spațiu, reprezentând coordonatele punctului de plecare, capăt al unui segment;

Linia M+N+3: x_D y_D
- două numere întregi, separate printr-un spațiu, reprezentând coordonatele punctului în care se află teroriștii, capăt al unui segment.

Date de ieșire

Fișier de ieșire: **BOND.OUT**

Linia 1: TMIN
- număr real pozitiv, reprezentând timpul minim în care James Bond ajunge la teroriști;

Liniile 2...: c_1 c_2
- pe următoarele linii, până la sfârșitul fișierului, se vor scrie câte două numere întregi, separate printr-un spațiu, reprezentând coordonatele capetelor de segmente prin care va trece James Bond pentru a ajunge la grupul de teroriști.

Restricții

- $1 \le M \le 100$
- $0 \le N \le 50$
- $1 \le R \le 30000$
- $0 \le T \le 30000$
- coordonatele capetelor segmentelor și ale pozițiilor bombelor aparțin intervalului [-30000, 30000].

Exemplu

BOND.IN
```
3 1
0 0 10 0
0 0 10 10
10 0 10 10
6 4 2 1
0 0
10 10
```

BOND.OUT
```
20
0 0
10 0
10 10
```

Observații

- numerele din fișierul de ieșire se vor afișa cu o precizie de două zecimale;
- se garantează existența soluției;
- datele de intrare sunt corecte.

Timp de execuție: 1 secundă/test

P050124: Șiruri

Radu Ștefan, Brașov

Se dau două șiruri având ambele câte N numere naturale, cuprinse între 1 și $N/2$ (N este par). Fiecare număr între 1 și $N/2$ apare de exact două ori în fiecare dintre cele două șiruri.

Cerință

Determinați subșirul comun având lungimea maximă.

Date de intrare

Fișier de intrare: **SIRURI.IN**

Linia 1: N
- număr natural nenul având semnificația din enunț;

Linia 2: a_1 a_2 a_3 ... a_N
- N numere naturale nenule, separate prin câte un spațiu, reprezentând elementele primului șir;

Linia 3: b_1 b_2 b_3 ... b_N
- N numere naturale nenule, separate prin câte un spațiu, reprezentând elementele celui de-al doilea șir.

Date de ieșire

Fișier de ieșire: **SIRURI.OUT**

Linia 1: NR
- numărul de elemente ale subșirului comun;

Linia 2: c_1 c_2 c_3 ... c_{NR}
- NR numere naturale nenule, reprezentând subșirul comun de dimensiune maximă.

Restricții

- $0 < N \le 15000$
- N este număr par

Exemplu

SIRURI.IN
```
10
3 5 4 4 2 5 2 3 1 1
1 4 2 2 3 4 5 5 3 1
```

SIRURI.OUT
```
5
4 2 2 3 1
```

Timp de execuție: 1 secundă/test

CONCURSUL de programare AGORA

S-a încheiat şi ultima rundă a Concursului de Programare Agora. În cadrul acestei număr vă vom prezenta soluţiile problemelor de la runda a şasea, clasamentul final al concursului, precum şi detalii referitoare la etapa finală la care participă primii zece clasaţi.

Cele trei probleme ale ultimei runde au fost relativ dificile, dovadă fiind numărul mic de concurenţi care au reuşit să obţină puncte. De obicei, vă prezentam şi un clasament al rundei care conţinea numele şi punctajele celor mai buni 50 de concurenţi la runda respectivă. De această dată am avut doar 49 de participanţi al căror punctaj să fie diferit de 0, motiv pentru care ne limităm la prezentarea celor care au obţinut primele 25 de punctaje. Aceştia sunt:

1. Mugurel Ionuţ Andreica, Bucureşti	*300*
1. Cosmin-Silvestru Negruseri, Bistriţa	*300*
3. Maximilian Machedon, Bucureşti	*275*
4. Liviu Lalescu, Craiova	*243*
5. Mihai Pătraşcu, Craiova	*233*
6. Radu Vătavu, Suceava	*217*
7. Vencel Bors, Oradea	*216*
8. Csaba András, Oradea	*213*
9. Valentin Bişa, Lugoj	*211*
10. Ciprian Baicu, Drobeta Turnu Severin	*200*
10. Adrian Cârcu, Bistriţa	*200*
10. George Drumea, Bucureşti	*200*
10. Octavian-Daniel Dumitran, Bucureşti	*200*
10. Flaviu Paşca, Bistriţa	*200*
10. Mihai Stroe, Bucureşti	*200*
10. Alexandru-Emilian Susu, Bucureşti	*200*
17. Csaba Pătcaş, Oradea	*189*
18. Dan Ghinea, Bucureşti	*186*
19. Valentin Pop, Carei	*172*
20. Adrian Balasko, Zalău	*164*
21. Ciprian Cană, Vaslui	*152*
22. Cristian Toth, Lugoj	*151*
23. Horea Coroiu, Cluj-Napoca	*143*
24. Bogdan Nicolae, Sibiu	*137*
25. Claudiu Gruia, Bucureşti	*122*

Despre probleme

Problemele de la ultima rundă au fost puţin mai dificile decât cele de la primele cinci.

Prima dintre ele era foarte simplă, fiind un exemplu clasic de aplicare a metodei *divide et impera*.

Cea de-a doua a fost, probabil, cea mai dificilă problemă propusă spre rezolvare în cadrul *Concursului de Programare Agora*. Era destul de greu de găsit algoritmul de rezolvare, dar cea mai dificilă parte a rezolvării o reprezenta implementarea eficientă a acesteia.

Ultima problemă a fost mai simplă. Cea mai "comodă" rezolvare a sa consta în folosirea "metodei constantelor".

Etapa finală

Primii zece clasaţi în urma celor şase etape ale concursului vor putea participa la faza finală care se va desfăşura la Cluj-Napoca pe data de 26 mai 2001.

La această etapă, pot participa ca invitaţi (în afara concursului) şi alţi participanţi la concursul organizat de revista noastră. Aceştia vor trebui să suporte cheltuielile de cazare şi transport şi, evident, trebuie să înştiinţeze organizatorii că doresc să se deplaseze la *Cluj*.

Bilanţ

După încheierea concursurilor se realizează, de obicei, bilanţuri. Vă prezentăm câteva date statistice referitoare la *Concursul de Programare Agora*.

Au fost 402 participanţi înscrişi, dar doar 176 dintre ei au reuşit să obţină puncte. Printre concurenţi s-au numărat şi 86 de elevi sau studenţi din afara României. La concurs au participat doar 22 de fete şi numai 12 dintre ele au obţinut puncte. 132 de concurenţi au reuşit să rezolve corect cel puţin o problemă şi 46 dintre ei au obţinut punctaj maxim (300 de puncte) la cel puţin una dintre runde.

Inversiuni

Această problemă a fost rezolvată corect de către 28 dintre cei care au participat la a șasea rundă a *Concursului de Programare Agora*. Problema era foarte cunoscută, ea fiind propusă spre rezolvare la numeroase alte concursuri.

Cea mai simplă idee de rezolvare este de a lua în considerare toate perechile (i, j) pentru care $i < j$ și a verifica dacă $a_i > a_j$.

Este evident că această metodă folosește $n \cdot (n - 1) / 2$ comparări, deci ordinul de complexitate al algoritmului este $O(n^2)$.

Chiar și un program care implementa un astfel de algoritm s-ar fi încadrat în limita de timp admisă dacă nu efectua operații suplimentare inutile. Au existat câțiva concurenți care au obținut 100 de puncte folosind această metodă de rezolvare.

Există și un algoritm de rezolvare, bazat pe metoda *divide et impera*, care are ordinul de complexitate $O(n \log n)$.

Etapa divide

Se împarte șirul de numere în două subșiruri de lungimi aproximativ egale. Primul subșir va conține prima jumătate a numerelor, iar al doilea cea de-a doua jumătate.

Etapa stăpânește

Se apelează recursiv algoritmul pentru cele două subșiruri formate și se determină numărul de inversiuni din fiecare dintre ele.

Etapa combină

Numărul de inversiuni din șir va fi egal cu suma numărului de inversiuni din cele două subșiruri la care se adaugă numărul de inversini formate de elemente din primul subșir cu elemente din al doilea subșir.

Pentru fiecare element din al doilea subșir va trebui să determinăm numărul de elemente din primul subșir care sunt mai mari decât el. Pentru aceasta vom ordona cele două subșiruri (această operație poate fi realizată pe parcursul apelurilor recursive succesive) și vom realiza o simplă operație de interclasare. În momentul în care vom adăuga la șirul interclasat un element din al doilea subșir, vom ști câte elemente mai mari decât el se află în primul subșir.

Practic, folosim algoritmul *mergesort* de ordonare a unui șir de numere naturale. Singura operație suplimentară pe care o efectuăm este numărarea inversiunilor.

Versiunea oficială a rezolvării acestei probleme este următoarea:

Au rezolvat corect:

Csaba András, Mugurel Ionuț Andreica, Ciprian Baicu, Adrian Balasko, Valentin Bișa, Vencel Bors, Lucian Bucur, Ciprian Cană, Adrian Cârcu, Horea Coroiu, George Drumea, Daniel Dumitran, Claudiu Gruia, Mihai Hărănguș, Alexandru Ivan, Liviu Lalescu, Maximilian Machedon, Mihail-Cosmin Pițu-Rada, Silvestru Negruseri, Bogdan Nicolae, Flaviu Pașca, Pătcaș Csaba, Mihai Pătrașcu, Lucian-Daniel Stanciu, Mihai Stroe, Alexandru Susu, Cristian Toth, Radu Vătavu

Listing INVERS.CPP

```
#include <stdio.h>

long n,nr,huge a[10001],huge b[10001];

void ReadData(void){
  FILE *f=fopen("INVERS.IN","rt");
  fscanf(f,"%d",&n);
  for (int i=0;i<n;i++)
    fscanf(f,"%ld",&a[i]);
  fclose(f);
}

void MergeSort(int x, int y){
  if (x==y) return;
  int m=(x+y)/2,i=x,j=m+1,p=0;
  MergeSort(x,m);
  MergeSort(m+1,y);
  while (i<=m || j<=y)
    if ((a[j]<a[i] && j<=y) || i>m)
      b[x+p++]=a[j++],nr+=m-i+1;
    else b[x+p++]=a[i++];
  for (i=x;i<=y;i++) a[i]=b[i],b[i]=0;
}

void WriteSolution(void){
  FILE *f=fopen("INVERS.OUT","wt");
  fprintf(f,"%ld",nr);
  fclose(f);
}

void main(void){
  ReadData();
  MergeSort(0,n-1);
  WriteSolution();
}
```

Runda 6, problema 2

Rețea

Această problemă a fost rezolvată corect doar de doi participanți la cea de-a șasea rundă a *Concursului de Programare Agora* și anume *Cosmin-Silvestru Negruseri* din *Bistrița* și *Mugurel Ionuț Andreica* din *București*. Au mai fost concurenți care au rezolvat corect problema, dar nu au respectat formatul fișierului de ieșire. Cea mai frecventă eroare care a apărut a fost separarea coordonatei orizontale de cea verticală prin virgulă în loc de spațiu.

Dintre cei 43 de concurenți care au trimis o soluție pentru această problemă doar șase au obținut peste 50 de puncte, iar 18 dintre ei nu au obținut nici un punct.

Problema se reduce la determinarea unui flux maxim într-o rețea de transport în care nodul sursă este conectat la punctele din care pornesc drumurile, iar nodul destinație este conectat la punctele de pe marginea rețelei.

Există și alte soluții echivalente care folosesc diferite metode de rezolvare. Totuși, în principiu, acești algoritmi de rezolvare a acestei probleme sunt echivalenți cu algoritmul de determinare a fluxului maxim într-o rețea de transport.

Vă prezentăm în continuare soluția propusă de *Cosmin-Silvestru Negruseri*, concurentul clasat pe primul loc după cea de-a șasea etapă a concursului nostru.

Listing NET.PAS

```
Program Retea;

type  matrix=array[0..101,0..101] of Byte;
      imatrix=array[0..101,0..101] of Integer;
      isir=array[1..20000] of Integer;
      sir=array[1..20000] of Byte;

const  dir:array[1..4,1..2] of Shortint=
                  ((-1,0),(0,-1),(1,0),(0,1));
       invers:array[1..4] of Shortint=
                                 (3,4,1,2);

var a:array[0..4] of matrix;
    at:array[1..2] of ^imatrix;
    vv:array[1..10,1..10] of Byte;
    x,y,z:^sir;
    t:^isir;
    lx,ly,p:array[1..400] of Byte;
    n,m,h,t1,t2,i,j,last,aux,pas:Longint;
    bool:Boolean;
    fis:Text;

procedure finish;
begin
  Assign(fis,'NET.OUT');
  Rewrite(fis);
  Writeln(fis,0);
  Close(fis);
  Halt
end;
function min(a1,a2:Integer):Integer;
begin
  if a1>a2 then
    min:=a2
  else min:=a1
end;

procedure qsort(l,r:Integer);
var i,j,mid:Integer;
begin
  i:=l;
  j:=r;
  mid:=p[(l+r) shr 1];
  repeat
    while p[i]<mid do
      Inc(i);
    while p[j]>mid do
      Dec(j);
    if i<=j then
    begin
      aux:=p[i];
      p[i]:=p[j];
      p[j]:=aux;
      aux:=lx[i];
      lx[i]:=lx[j];
      lx[j]:=aux;
      aux:=ly[i];
      ly[i]:=ly[j];
      ly[j]:=aux;
      Inc(i);
      Dec(j)
    end
  until i>j;
```

```
  if l<j then
    qsort(l,j);
  if i<r then
    qsort(i,r)
end;

procedure readdata;
begin
  Assign(fis,'NET.IN');
  Reset(fis);
  Readln(fis,n,m);
  if m>4*n-4 then
  begin
    Close(fis);
    finish
  end;
  for i:=1 to m do
    Readln(fis,lx[i],ly[i]);
  Close(fis);
  for i:=1 to m do
    p[i]:=min(lx[i]-1,min(ly[i]-1,
                  min(n-lx[i],n-ly[i])));
  qsort(1,m)
end;

procedure add(v,ii,jj,kk:Integer);
begin
  if (ii<1) or (ii>n) or (jj<1) or (jj>n) or
                  (at[kk]^[ii,jj]=pas) then
    Exit;
  at[kk]^[ii,jj]:=pas;
  inc(t2);
  x^[t2]:=ii;
  y^[t2]:=jj;
  z^[t2]:=kk;
  t^[t2]:=v;
  if (ii=1) or (jj=1) or (ii=n) or (jj=n)
    then bool:=true
end;

procedure expand(v,i,j,k:Integer);
begin
  if k=1 then
  begin
    if a[0,i,j]=0 then
      add(v,i,j,2);
    if a[3,i-1,j]=1 then
      add(v,i-1,j,2);
    if a[1,i+1,j]=1 then
      add(v,i+1,j,2);
    if a[4,i,j-1]=1 then
      add(v,i,j-1,2);
    if a[2,i,j+1]=1 then
      add(v,i,j+1,2)
  end
  else
    begin
      if a[3,i,j]=0 then
      begin
        add(v,i+1,j,1);
        if bool then
          Exit
      end;
      if a[1,i,j]=0 then
      begin
        add(v,i-1,j,1);
        if bool then
          Exit
      end;
      if a[4,i,j]=0 then
      begin
        add(v,i,j+1,1);
        if bool then
          Exit
      end;
      if a[2,i,j]=0 then
      begin
        add(v,i,j-1,1);
        if bool then
          Exit
      end;
      if a[0,i,j]=1 then
        add(v,i,j,1)
    end
end;

procedure lee_it(ii,jj:Byte);
var i:Integer;
begin
  bool:=false;
  h:=1;
  t1:=1;
  x^[1]:=ii;
  y^[1]:=jj;
  z^[1]:=1;
  t^[1]:=0;
  repeat
    t2:=t1;
    for i:=h to t1 do
    begin
      expand(i,x^[i],y^[i],z^[i]);
      if bool then
        Break
    end;
    if bool then
      Break;
    h:=t1+1;
    t1:=t2;
  until h>t1
end;
```

```
procedure drum(vvv:Integer);
var i,j,i1,j1,dire,v:Integer;
begin
  v:=vvv;
  i:=x^[v];
  j:=y^[v];
  while t^[v]<>0 do
  begin
    v:=t^[v];
    i1:=i;
    j1:=j;
    i:=x^[v];
    j:=y^[v];
    if (i=i1) and (j=j1) then
      if a[0,i,j]=1
        then a[0,i,j]:=0
      else a[0,i,j]:=1;
    else
      begin
        for dire:=1 to 4 do
          if (i+dir[dire,1]=i1) and
                    (j+dir[dire,2]=j1) then
            Break;
        if a[invers[dire],i1,j1]=1 then
          a[invers[dire],i1,j1]:=0
        else
          begin
            a[dire,i,j]:=1;
            a[invers[dire],i1,j1]:=0
          end
      end
  end
end;
```

```
procedure work;
var i,j:Integer;
begin
  New(x);
  New(y);
  New(z);
  New(t);
  New(at[1]);
  New(at[2]);
  Fillchar(at[1]^,Sizeof(at[1]^),0);
  Fillchar(at[2]^,Sizeof(at[2]^),0);
  for pas:=1 to m do
    if (lx[pas]<>1) and (lx[pas]<>n) and
          (ly[pas]<>1) and (ly[pas]<>n) then
    begin
      for j:=1 to m do
        at[1]^[lx[j],ly[j]]:=pas;
      lee_it(lx[pas],ly[pas]);
      if h>t1 then finish;
      drum(t2)
    end;
  last:=pas;
  Dispose(x);
  Dispose(y);
  Dispose(z);
  Dispose(t);
  Dispose(at[1]);
  Dispose(at[2])
end;
```

```
procedure writedata;
var k,i,j,nr,k1:Integer;
begin
  Assign(fis,'NET.OUT');
  Rewrite(fis);
  for k:=1 to m do
  begin
    nr:=1;
    i:=lx[k];
    j:=ly[k];
    while a[0,i,j]=1 do
    begin
      for k1:=1 to 4 do
        if a[k1,i,j]=1 then
        begin
          i:=i+dir[k1,1];
          j:=j+dir[k1,2];
          Break
        end;
      Inc(nr)
    end;
    Writeln(fis,nr);
    i:=lx[k];
    j:=ly[k];
    Write(fis,'(',i,',',j,')');
    while a[0,i,j]=1 do
    begin
      for k1:=1 to 4 do
        if a[k1,i,j]=1 then
        begin
          i:=i+dir[k1,1];
          j:=j+dir[k1,2];
          Write(fis,'(',i,',',j,')');
          Break
        end;
    end;
    Writeln(fis)
  end;
  Close(fis)
end;

Begin
  readdata;
  work;
  writedata
End.
```

Numere prime

Această problemă a fost rezolvată corect de 17 dintre cei care au participat la a șasea rundă a *Concursului de Programare Agora*. Problema era dificilă dacă se încerca scrierea unui algoritm de verificare a primalității numerelor, dar relativ simplă dacă se folosea *metoda constantelor*.

Vom furniza ca soluții cele mai mici numere de n cifre care sunt prime. Acestea pot fi obținute prin diferite metode. Trebuie remarcat faptul că aceste numere au forma $10^{n-1} + k$, unde k este o valoarea relativ mică.

Se știe că există aproximativ $ln\ n$ numere prime mai mici decât un număr n.

Este evident că există aproximativ $(n - 1) \cdot ln\ 10$ numere prime mai mici decât 10^{n-1}.

Vom încerca să aproximăm valoarea maximă k în cel mai defavorabil caz (100 de numere prime cu 100 de cifre).

Numărul de numere prime cuprinse între n și $n + k$ este de aproximativ $k / ln\ n$.

Putem deduce destul de ușor că valoarea maximă a lui k este de aproximativ $10000 \cdot ln\ 10$.

Așadar valorile k pot fi stocate folosind constante de patru bytes.

Puteau fi scrise și programe care memorau în întregime numerele prime care constituiau soluțiile. Apar probleme datorită numărului mare de constante care trebuie memorate. Din această cauză codul sursă depășea dimensiunea de 64 KB. Mediul *Borland Pascal 7.0* nu permite compilarea programelor care depășesc această dimensiune, dar mediul *Borland C++ 3.1* poate compila și astfel de surse dacă sunt specificate opțiunile de compilare adecvate.

Versiunea oficială a rezolvării acestei probleme are următoarea formă.

Listing PRIME.CPP

```
#include <stdio.h>

const long
  digits_2[]={1,3},
  digits_3[]={1,3,7},
  ...

void main(void){
  FILE *f=fopen("PRIME.IN","rt");
  int i,m,n;
  fscanf(f,"%d%d",&n,&m);
  fclose(f);
  f=fopen("PRIME.OUT","wt");
  switch (n){
    case 1: fprintf(f,"3\n");
            break;
    case 2: for (i=0;i<m;i++)
              fprintf(f,"1%01ld\n",
                                digits_2[i]);
            break;
    case 3: for (i=0;i<m;i++)
              fprintf(f,"1%02ld\n",
                                digits_3[i]);
            break;
    ...
    case 100: for (i=0;i<m;i++)
                fprintf(f,"1%0991d\n",
                                digits_100[i]);
  }
  fclose(f);
}
```

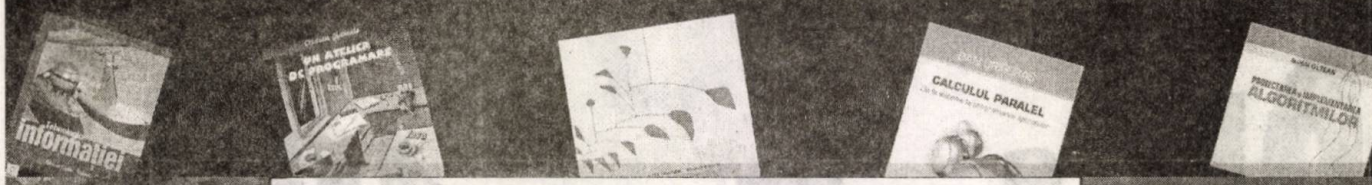
Au rezolvat corect:

Csaba András, Oradea
Mugurel Ionuț Andreica, București
Ciprian Baicu, Drobeta Turnu Severin
Bors Vencel, Oradea
Adrian Cârcu, Bistrița
George Drumea, București
Octavian-Daniel Dumitran, București
Dan Ghinea, București
Liviu Lalescu, Craiova
Maximilian Machedon, București
Cosmin-Silvestru Negruseri, Bistrița
Flaviu Pașca, Bistrița
Mihai Pătrașcu, Craiova
Valentin Pop, Carei
Mihai Stroe, București
Alexandru-Emilian Susu, București
Radu Ștefan, Brașov

Clnfo Top 100

1. Silvestru Negruseri, Bistrița	1709	6. Csaba András, Oradea	1612
2. Maximilian Machedon, București	1675	7. Ciprian Cană, Vaslui	1601
3. George Drumea, București	1665	8. Mihai Stroe, București	1589
4. Mugurel Andreica, București	1629	9. Liviu Lalescu, Craiova	1571
5. Adrian Cârcu, Bistrița	1614	10. Radu Ștefan, Brașov	1524

11. Ciprian Baicu, Drobeta Turnu Severin	1506	41. M.A. Safari Ghahsareh, Iran	1020	71. Andrei Markovits, Satu Mare	575
12. Dan Ghinea, București	1499	42. Daniel Crișan, Râmnicu Vâlcea	1015	72. Livia Bana, Drobeta Turnu Severin	562
13. Mihai Pătrașcu, Craiova	1479	43. Horea Coroiu, Cluj-Napoca	993	73. Laurent Demonet, Franța	520
14. Daniel Dumitran, București	1414	44. Valentin Pop, Carei	992	74. Mihai Stoicescu, Boldești-Scăeni	487
15. Bors Vencel, Oradea	1345	45. Sorin Otescu, Brașov	984	75. Ștefan Bucur, București	478
16. Radu Vătavu, Suceava	1309	46. Vlad Dascălu, Bacău	947	76. Daniel Bolohan, Iași	469
17. Claudiu Gruia, București	1277	47. Liviu-Cosmin Andreicuț, București	905	77. Aurel-Mihai Fulger, Constanța	460
18. Cristian Băicoianu, Ploiești	1256	48. Kuczapski Artur, Oradea	891	78. Constantin Asofiei, Târgu Neamț	457
19. Gabriel Pârvan, Râmnicu Sărat	1237	49. Marius Andrei, București	886	79. Bogdan Chirilă, Botoșani	456
20. Vlad Vâlceanu, Arad	1218	50. Andrei Giurgiu, București	880	80. Silviu-Gabriel Udrea, Târgu Jiu	451
21. Liviu Păunescu, Constanța	1216	51. Andrei David, Râmnicu Vâlcea	878	81. Raluca Mușăloiu-Elefteri, București	443
22. Flaviu Pașca, Bistrița	1211	52. Codruț-Lucian Lazăr, Câmpeni	861	82. Mihai Gojol, Cluj-Napoca	438
23. Denis Bogdănaș, Iași	1204	53. Mihail Pit-Rada, Dr. Tr. Severin	858	83. Petru-Simon Moț, Brăila	437
24. Pătcaș Csaba, Oradea	1193	54. Victor Vernescu, Constanța	855	84. V. Muthuramakrishnan, India	421
25. Cristian Toth, Lugoj	1185	55. Ioana-Maria Ileană, Alba Iulia	824	85. Nadina Busuioc, Constanța	415
26. Cristian Alexandrescu, Botoșani	1176	56. Aurelian Ghiță, Buzău	785	86. Andrei-Liviu Dumbrava, Iași	414
27. Bogdan Stan, Câmpina	1165	57. Balasko Adrian, Zalău	768	87. Tiberius Pârcălabu, București	412
28. Mihai-Vlad Pantiș, Cluj-Napoca	1159	58. Lucian-Raul Silistru, Vaslui	757	88. Elvis Asaftei, Vaslui	400
29. Valentin Bișa, Lugoj	1158	59. Arcadie Crăcan, Iași	746	89. Johannes Slotta, Germania	398
30. Victor Asavei, Drobeta Turnu Severin	1147	60. Cosmin Răianu, Constanța	745	90. Paul Rusu, Cluj-Napoca	396
31. Bogdan Nicolae, Sibiu	1144	61. Răzvan Mușăloiu-Elefteri, București	706	91. Radu Cocieru, Sibiu	367
32. Alexandra-Elena Constantin, Ploiești	1134	62. Cătălin Kesa, Bușteni	696	92. Petko Minkov, Bulgaria	366
33. Bogdan Dumitru, Ploiești	1111	63. Paul Marinescu, Buzău	693	93. Bogdan-Vlad Țira, Cluj-Napoca	349
34. Horia Ciurdar, Timișoara	1056	64. Bogdan-Milovan Piloga, Timișoara	689	94. Emilian Miron, Bacău	337
34. Vincent Groenhuis, Olanda	1056	65. Adrian Cearnău, București	672	95. Raluca Sauciuc, București	330
36. Alexandru Moșoi, Bacău	1055	66. Andrei Benea, Focșani	657	96. Lucian Burja, Blaj	320
37. Mihail Popa, București	1038	67. Cristian Bojinovici, București	644	97. Alexandru Ivan, București	316
38. Mihai Hărănguș, Cluj-Napoca	1037	68. Lucian-Daniel Stanciu, Brăila	618	97. Costin Speciac, Buzău	316
39. Dan Popovici, Arad	1035	69. Radu Dondera, București	617	99. Perini Arpad, Sfântu Gheorghe	312
40. Andrei Csibi, Cluj-Napoca	1026	70. Bogdan Lucaciu, Dr. Turnu Severin	612	100. Dan-Octavian Savu, București	305



COMPUTER LIBRIS AGORA

Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS
AGORA

C și LINUX în Finlanda

Mihai Pătrașcu

Începând de anul acesta, sistemul de operare folosit la olimpiadele internaționale de informatică va fi Linux. Limbajul C s-a impus de mult timp în lumea Unix și este, fără nici un dubiu, cel mai folosit limbaj sub aceste sisteme. Vom încerca să prezentăm câteva elemente pentru a ajuta tranziția programatorilor în C de la mediile Borland la uneltele GNU/Linux.

Mediul de programare

Varietatea de editoare și medii de programare existente ne împiedică să recomandăm unul anume. Fiecare programator are un mediu preferat și va găsi numeroase motive pentru a-și motiva alegerea. Personal, autorul folosește de mai mulți ani editorul `joe` și este foarte mulțumit de performanțele lui.

Puține editoare oferă și facilități de compilare a programelor, deci ar putea fi util să vă obișnuiți cu folosirea mai multor terminale. Deschizând două terminale, puteți folosi editorul dumneavoastră favorit în unul dintre acestea și *shell*-ul *bash* în cel de-al doilea, care va fi folosit pentru compilarea, rularea și depanarea programelor.

Să presupunem că ați creat fișierul **`sum.c`** care conține codul:

```
#include <stdio.h>
int main(){
  int a,b;
  fscanf(fopen("sum.in","r"),"%d%d",&a,&b);
  fprintf(fopen("sum.out","w"),"%d\n",a+b);
  return(0);
}
```

Puteți compila și testa acest program folosind comenzile:

```
% gcc sum.c -o sum -Wall -O2
% echo 3 7 > sum.in
% time sum
...      (timpii de rulare ai programului)
% cat sum.out
10
```

Caracterul `%` este *prompt*-ul afișat de *shell* și poate varia. La unele distribuții directorul curent nu este inclus în variabila de mediu `PATH` și este necesar să tastăm `time ./sum` pentru a rula programul. Sunt raportați trei timpi de rulare: ***timpul real*** (*wall-clock* time), ***timpul efectiv*** de rulare al programului (fiind vorba de un sistem *multitasking*, pot fi rulate și alte programe simultan, deci acest timp este mai mic decât cel real), și ***timpul logic*** care reprezintă timpul consumat efectiv de program, la care se adaugă timpul consumat de sistemul de operare pentru a efectua operații cerute de programul respectiv (de exemplu, pentru a scrie rezultatele pe disc). Pe un sistem pe care nu rulează nici un alt program, timpul fizic este egal cu cel logic (aceasta va fi probabil situația în timpul evaluării soluțiilor prezentate de concurenți la *IOI 2001*).

O greșeală frecventă la începători este să denumească sursa **`test.c`** și să creeze executabilul **`test`**. Încercarea de a rula acest executabil va da rezultate neașteptate pentru că există deja un utilitar *shell* cu acest nume și acesta va fi executat datorită ordinii directoarelor în `PATH`.

Pentru compilarea programelor se recomandă specificarea opțiunilor `-Wall -O2`, pentru afișarea tuturor avertismentelor generate de compilator și optimizarea codului generat. Dacă omitem opțiunea `-O2`, pe lângă generarea unui cod mai puțin eficient, mai există un efect secundar nedorit: unele posibile greșeli nu vor fi depistate și raportate, deși am specificat `-Wall`.

Programarea *shell* se poate dovedi a fi un atu în unele cazuri. Acesta este însă un domeniu prea vast pentru a-l descrie amănunțit în cadrul acestui articol. Ne vom opri totuși asupra unui utilitar: `bc`. Acesta oferă un limbaj script asemănător limbajului C, cea mai importantă facilitate fiind reprezentată de faptul că întregii pot fi oricât de mari, mărimea lor fiind limitată doar de memoria disponibilă. Acest utilitar poate fi un ajutor neprețuit în testarea programelor care folosesc numere mari. Să presupunem că dorim să calculăm valoarea `100!`. Cea mai comodă metodă este să tastăm:

```
% bc
for(fact=i=1;i<=100;i++) fact*=i;
```

```
fact
...        (bc afișează valoarea 100!)
quit
```

Observați că afișarea valorii unei expresii, în particular o variabilă, se face scriind, pur și simplu, acea expresie.

Depanarea

Puține programe funcționează perfect la prima rulare. Mai devreme sau mai târziu va trebui să depanăm un program care funcționează conform așteptărilor. Deși există unele medii de dezvoltare care au unele facilități de depanare, vom constata că există soluții foarte bune și fără a recurge la acestea.

Cea mai simplă și mai folosită metodă de depanare este modificarea programului pentru a afișa unele informații esențiale în anumite puncte critice. Astfel, citind ieșirea programului vom putea analiza evoluția sa. Dublată de puțină experiență, această abordare se dovedește foarte eficientă.

Deși există posibilitatea afișării informațiilor de depanare la ieșirea standard, este mult mai recomandabil să se folosească ieșirea de eroare pentru acest lucru (`stderr`). Un motiv este că cel mai comod mod de a lucra cu fișiere în timpul concursurilor este să redirectăm intrarea și ieșirea standard la fișierele specificate de problemă. Multe soluții conțin o secvență asemănătoare cu următoarea:

```
freopen("prob2.in","r",stdin);
scanf("%d",&n);
...
freopen("prob2.out","w",stdout);
printf("%d\n",min);
...
```

Un motiv mult mai puternic este că în cazul terminării subite a programului dumneavoastră (cum ar fi accesul la o adresă de memorie invalidă), mesajele scrise la ieșirea standard pot să nu fie trimise sistemului de operare și nu vor fi niciodată afișate.

Pentru a micșora numărul de caractere tastate pentru afișarea mesajelor la `stderr`, ne putem folosi de o extensie a compilatorului `gcc`:

```
#define pe(fmt,args...) \
fprintf(stderr, fmt ,##args)
...
pe("<i=%d,min=%d>",i,min);
```

Trebuie să fiți atenți la dispunerea spațiilor în jurul virgulei care precede cele două diezuri.

O altă variantă des folosită, care este conformă standardului *ANSI C*, este următoarea:

```
#define pr(x) fprintf(stderr, #x " = %d\n", x)
...
pr(n * k - i);
```

Presupunând că valoarea expresiei este 13, programul va scrie la ieșirea de eroare:

```
n * k - i = 13
```

O problemă o constituie programele care scriu o cantitate foarte mare de informații la ieșirea de eroare (uneori și compilatorul `gcc` generează sute de erori). În mod normal, un program de paginare ca `less` nu citește decât ieșirea standard. Îl putem face să citească și ieșirea de eroare prin următoarea secvență:

```
$ prob3 2>/dev/stdout | less
```

Standardul *ANSI C* prevede două facilități care ajută la depanare. Funcția `abort()` încheie execuția programului și generează un fișier *core*. Puteți apela această funcție când depistați un caz "imposibil". Astfel, când se întâmplă un eveniment neprevăzut, programul dumneavoastră se va opri și veți putea examina starea memoriei în acel moment, putând constata unde ați greșit în raționamentul dumneavoastră. Celălalt ajutor în depanare este funcția `assert()`. Acesta primește un parametru, de obicei o condiție care sunteți siguri că este îndeplinită. Dacă condiția nu este îndeplinită, se comportă asemănător cu `abort()`.

Dacă aveți experiență în programarea sub Linux, probabil vi s-a întâmplat ca programul dumneavoastră să se oprească și să fie generat un fișier *core*. Cea mai frecventă cauză este semnalul `SIGSEGV`, care este generat de încercarea programului de a accesa o zonă invalidă de memorie: accesarea unui pointer cu valoarea `NULL`, folosirea unui pointer neinițializat (care conține o valoare aleatoare) sau accesarea unui element de tablou folosind un indice invalid.

Să considerăm următorul exemplu:

```
% prob3
Segmentation fault (core dumped)
% gdb prob3 core
Program terminated with signal 11,
Segmentation fault.
#0  0x804839e in main () at prob3.c:7
7                   a[x[i] - k] = 1;
(gdb) print i
$1: 17
(gdb) print x[i]-k
$2: 17774212
```

Este evident că vectorul `a[]` a fost indexat cu o valoare incorectă. Mai trebuie precizat că unele distribuții setează o opțiune care împiedică generarea fișierelor *core*. În acest caz puteți folosi următoarea comandă:

```
$ ulimit -c 65536
```

Am făcut între timp cunoștință cu depanatorul *GNU*: `gdb`. În afara examinării fișierelor *core*, este foarte comodă folosirea depanatorului pentru examinarea cursului execuției programului și a evoluției variabilelor pentru cazuri de complexitate redusă. Să presupunem că, pentru a fi siguri de corectitudinea programului, ne interesează valorile unui vector după execuția unei anumite funcții. Cea mai rapidă soluție este:

```
% gcc prob3.c -o prob3 -Wall -g
% gdb prob3
(gdb) break 113
(gdb) run
(gdb) print a
$1: { 7, 9, 3, 1, 0 <repeats 996 times> }
(gdb) quit
```

Pentru ca gdb să fie cât mai util trebuie să compilăm programul ca în prima linie din exemplu. După terminarea depanării, pentru obținerea executabilului final este recomandabil să compilăm programul cu opțiunile prezentate la începutul articolului.

Comanda break stabilește un punct de întrerupere la o anumită linie (în exemplu, 113) sau la prima linie a unei funcții (de exemplu: break main). Observăm de asemenea că print se descurcă foarte bine cu vectori, deci îl putem folosi pentru a afișa vectori oricât de mari, cu condiția ca numai un număr rezonabil de elemente să fie folosite.

Alte două comenzi utile sunt up și down. Acestea se deplasează în sus sau în jos pe stivă. Astfel, putem examina variabilele locale funcției care a apelat funcția curentă, sau funcției care a apelat-o la rândul ei pe aceasta etc.

Comenzile folosite pentru urmărirea cursului execuției programului sunt:

- continue - continuă execuția până la următorul punct de întrerupere;
- next - execută linia curentă;
- step - execută linia curentă, dar dacă acesta conține apelul unei funcții, "pășește" în interiorul funcției, oprindu-se la prima linie a acesteia;
- until - continuă execuția până când programul ajunge la o linie aflată după cea curentă. Este utilă spre exemplu pentru a trece peste o buclă neinteresantă. Poate primi și un argument, care este identic cu cel pentru break.

Comenzile anterioare sunt cele mai folosite, motiv pentru care este suficient să tastăm prima literă a comenzii în locul numelui complet.

Pentru a monitoriza constant evoluția unei valori, putem folosi comanda display. Aceasta este asemănătoare cu print, dar ea reevaluează și afișează o expresie după fiecare dintre aceste comenzi care execută secvențe din program. Mai trebuie menționat că gdb are o facilitate de *TAB-completion* foarte utilă, la fel ca și *bash*, iar informații suplimentare despre orice comandă se pot obține cu comanda help. Două comenzi utile nemenționate până acum sunt delete și undisplay, care anulează o comandă break sau display.

Rafinări ale limbajului C

Probabil cea mai evidentă schimbare față de mediile *Borland C* o reprezintă trecerea la întregi pe 32 de biți. A apărut și un tip nou, **long long**, care este un întreg pe 64 de biți. Compilatorul suportă și vectori locali de dimensiuni variabile:

```
int solve(int n){
  int a[n];
  ...
}
```

Această facilitate permite o scriere mai elegantă a codului și o economisire a memoriei (nu declarăm vectori având dimensiunile maxime posibile, ci vectori cu dimensiunile necesare). În timp ce folosirea acestei facilități nu are efecte negative pentru vectori (am întâlnit chiar programe care rulau semnificativ mai rapid când vectorii erau declarați astfel), ea duce la o oarecare încetinire a codului pentru matrice multidimensionale, deși de obicei diferența nu este semnificativă.

Programatorul trebuie să acorde atenție unui anumit aspect. Deși nu mai există restricțiile extrem de severe privitoare la memorie de sub *DOS*, comisia de evaluare stabilește unele restricții pentru memoria ce poate fi folosită de un program. Tendința este să se limiteze stiva la 1 MB de memorie, iar datele la 15 MB. Astfel, va trebui să avem grijă să nu încercăm să alocăm mai mult de 1 MB de spațiu pe stivă. Dacă este necesară mai multă memorie, tablourile de mari dimensiuni vor trebui declarate globale, fiind astfel mutate în zona de date, unde putem aloca 15 MB.

O facilitate importantă este atributul **inline** al funcțiilor. O funcție a cărei declarații este precedată de **inline** este integrată în corpul funcției apelante. Pentru funcții mici apelate des, aceasta poate duce la o sporire considerabilă a vitezei. Se poate împărți astfel programul în funcții fără restricții; se câștigă claritate și nu se pierde din viteză nici în cazul funcțiilor apelate des în bucle interioare.

Această facilitate trebuie folosită cu discernământ. Dacă încercăm să integrăm o funcție destul de complexă (cu multe variabile) într-o altă funcție complexă, codul va deveni mai lent. Acest lucru este datorat arhitecturii; procesoarele din generația *x86* au foarte puțini regiștri generali.

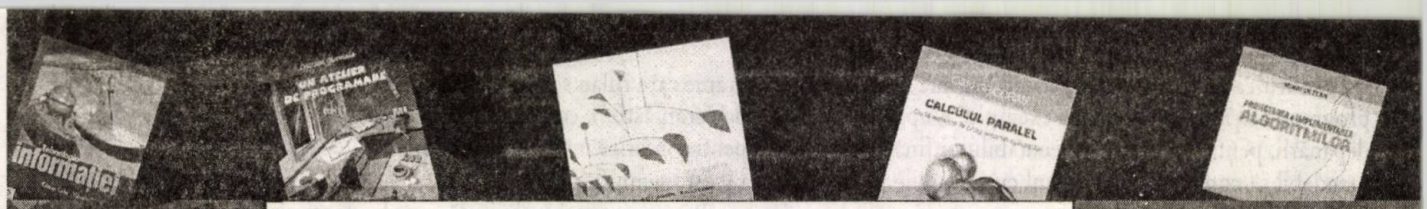
Vom discuta în continuare un aspect general valabil în orice mediu C, dar mai puțin cunoscute. În primul caz, problema este cauzată de faptul că limbajul C nu oferă decât vectori indexați începând cu zero. Mulți consideră acest fapt un dezavantaj, dar vom arăta că este doar un detaliu minor de sintaxă. Să presupunem că vrem să declarăm un vector care să fie indexabil cu valori între -100 și +150. Putem face asta foarte ușor astfel:

```
int a[251];
#define a              (a + 100)
```

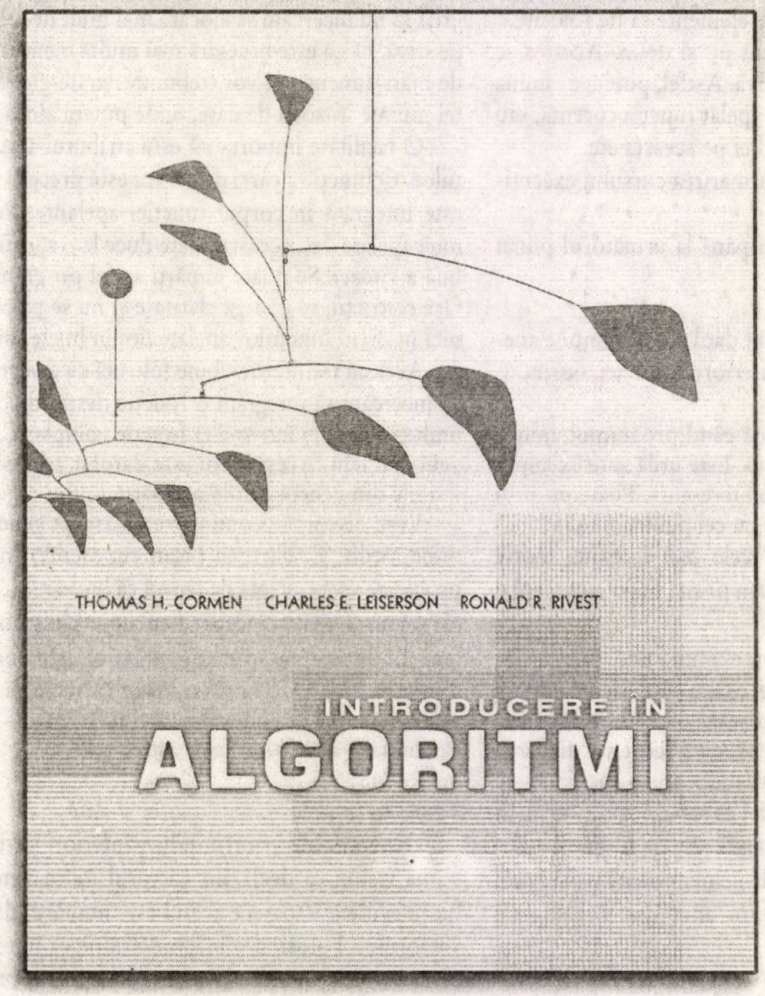
Generalizând pentru indicii inferiori și superiori *X* și *Y*, va trebui să declarăm vectorul de mărimea (*Y*-*X*+1), lucru evident deoarece acesta este numărul de elemente al vectorului. Folosind o macrodefiniție ca în exemplu, preprocesorul va substitui orice referință la a prin (a+100). Acesta este un pointer la elementul care dorim să-l adresăm ca elementul 0. Elementul inferior al vectorului este (a+100)[-100], deci fizic a[0], iar elementul superior (a+100)[150], adică a[250], care este chiar ultimul element al vectorului. Acest truc ne poate ușura munca foarte mult în unele cazuri.

Mulți consideră că *C+Unix* reprezintă mediul ideal pentru programare. Astfel trecerea olimpiadelor internaționale pe aceste sisteme este una perfect justificată. Recomandăm și elevilor să nu ezite în a renunța la mediile de dezvoltare care rulează sub *DOS* și să se bucure de multiplele avantaje oferite de mediul *GNU/Linux*.

Mihai Pătrașcu este elev în clasa a XII-a la Liceul Carol I din Craiova. poate fi contactat prin e-mail la adresa **mihai_p@pcnet.ro**.



COMPUTER LIBRIS AGORA



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Regulament IOI 2001

Olimpiada Internațională din 2001 se va desfășura între 14 și 21 iulie în Finlanda, la Tampere. Zilele de concurs vor fi 16 și 18, când între orele 9-14 concurenți din peste 70 de țări se vor confrunta cu câte trei probleme. Pentru a nu-i surprinde pur și simplu cu multele modificări pe participanți, organizatorii îi invită în 15 iulie la un antrenament.

Echipamente

IOI-ul în *Tampere* se va desfășura pe calculatoare *Osborne Pro PC* cu procesor *Intel Pentium III* la 933 MHz și 128 MB RAM. Evident, vor avea tastatură englezească, mouse și monitor color.

Concurenții nu vor intra în sala de concurs cu dischete, dispozitiv pentru efectuarea calculelor, materiale scrise sau tipărite, respectiv dispozitive de comunicare. Dacă totuși o fac, vor putea fi descalificați.

Mediul de programare

Pe calculatoare vor fi instalate două sisteme de operare (*dual-boot*): *Debian Linux* versiunea 2.2r2 și *Windows 98 SE*.

Din motive de securitate pot fi mici modificări față de instalările standard. Instrumentele soft vor fi instalate în partiția utilizatorului, astfel încât concurenții nu trebuie să efectueze setări suplimentare.

Ambele platforme vor include:

- Compilator `gcc` versiunea 2.95.2.
- Compilator *freepascal* (`fpc`) versiunea 1.0.4.

Acestea sunt cele două compilatoare oficiale ale IOI 2001.

Linux

Pachetele *Linux* instalate vor fi:

- Versiunea *Rhide* 1.4.7.8, care folosește compilatorul `gcc`.
- *Freepascal ide*, versiunea 0.9.1.
- *Gdb debugger*, versiunea 4.18.
- *Ddd* (*Data Display Debugger*) versiunea 3.2, cu *debugger* grafic.
- Editorul *Emacs*, versiunea 20.7.
- Editorul *Vim*, versiunea 5.6.
- *Binutils* versiunea 2.9.5.0: asamblor *GNU*, linkeditor și utilități binare.

Se vor instala manuale standard și pagini cu informații utile!

Windows

Mediul *Windows* va include următoarele componente:

- Versiunea *Rhide* 1.4.7.8, care folosește compilatorul `gcc`.
- *Freepascal ide*, versiunea 0.9.1.
- *Gdb debugger*, versiunea 5.0.

Instalarea *Windows* va include editorul standard *MS-DOS Edit* (versiunea 2.0.026) și *Microsoft Notepad* standard. `Gcc` va fi instalat ca o parte din pachetul *DJGPP*, împreună cu fișiere "info". *Rhide* de asemenea va fi instalat cu fișiere "info". *Freepascal* va fi instalat în versiune "full".

Instrumente suplimentare

Se vor pune la dispoziția concurenților instrumente suplimentare, menite să-i ajute în rezolvarea problemelor. Toate documentațiile care nu au relevanță în ceea ce privește natura problemelor, vor fi disponibile pe paginile *Web* ale concursului în luna mai.

Documentațiile vor include informații care se pot traduce din timp și se pot pune la dispoziția concurenților inclusiv în timpul concursului. Pot exista anumite mesaje sau alte materiale care vor putea fi traduse doar în timpul traducerii problemelor pe durata ședinței *Adunării Generale* a IOI.

Problemele de concurs

Toate problemele vor fi de algoritmică. Soluția pentru o problemă va putea fi:

- un fișier sursă, sau
- un număr oarecare de fișiere de date de ieșire, produse pe baza unor date de intrare, folosind calcule de natură algoritmică.

Fiecare problemă va avea un titlu și un nume scurt. Numele scurt folosește la identificarea problemei.

Documentațiile problemelor ale căror soluții se cer sub forma unor fișiere sursă vor conține:

- tipul parametrilor de intrare și de ieșire, respectiv limitele valorilor acestora;

- limitele resurselor (timp, memorie disponibilă);
- opțiunile de compilare folosite la evaluare;
- posibile restricții privind comportamentul funcțiilor (sursele) (de exemplu, este posibil să se interzică utilizarea unor fișiere auxiliare).

Tipul parametrilor de intrare și de ieșire:

- întregi;
- caractere;
- șiruri de caractere;
- tablouri, posibil multi-dimensionale, având elemente de tipurile de mai sus.

Documentațiile problemelor ale căror soluții se cer sub forma unor fișiere de date de ieșire vor conține:

- structura fișierelor de intrare și de ieșire,
- directorul unde datele de intrare sunt pe calculatorul concurentului.

În acest caz fișierele de intrare vor fi fișiere text *ASCII*. Datele de intrare vor consta într-o secvență de articole. Un articol este un șir de caractere tipăribile (diferite de spațiu) (codurile *ASCII* de la 33 la 126).

Aceste articole pot fi separate prin câte un spațiu sau prin caracterul de sfârșit de linie). Structura fișierului de intrare (formatul) se va specifica în descrierea problemei.

Fișierele de ieșire vor consta într-o secvență de articole, în structuri (formate) similare cu ale fișierelor de intrare și aranjate conform instrucțiunilor din descrierea problemei.

Atunci când concurenții își vor prezenta răspunsurile se va verifica formatul fișierului de ieșire și se vor trimite atenționări dacă formatul este incorect.

Concursul

Concursul propriu-zis va fi asemănător cu cele de până acum, totuși, vom menționa câteva noutăți.

Răspunsurile vor putea fi înaintate de către concurenți în timpul concursului. Concurenții își vor înainta răspunsurile lor către serviciul pus la dispoziție de mediul concursului. Serviciul respectiv va verifica anumite proprietăți ale răspunsurilor.

Dacă răspunsul înaintat este un fișier sursă, acest serviciu va compila această sursă și va executa programul pe un test simplu.

Dacă răspunsul constă într-un fișier de date de ieșire, serviciul va verifica formatul acestuia.

Dacă pe parcursul acestor verificări se descoperă erori, concurentul va fi anunțat. Dacă nu există asemenea erori, răspunsul va fi acceptat pentru evaluare și concurentul va fi informat despre acest lucru.

De altfel, evaluarea propriu-zisă se va realiza separat și concurenții nu vor fi informați despre punctaj în timpul concursului.

Dacă un concurent va trimite mai multe răspunsuri pentru aceeași problemă, în evaluarea propriu-zisă se va lua în considerare ultima care a trecut prin serviciul de verificare.

Evaluarea

Sistemul de verificare va evalua răspunsurile concurenților în mod automat imediat după ce aceste răspunsuri au sosit.

Formularele cu rezultate (borderourile) se vor lista imediat după ce concursul s-a încheiat, apoi se vor difuza delegațiilor. Contestațiile se pot înainta până la termenul care va fi anunțat de *Comitetul Științific*.

Pentru evaluarea propriu-zisă, sursele se vor recompila și se vor executa sub *Linux*. Dacă un concurent nu a primit un mesaj de eroare, acest lucru nu înseamnă că el va avea punctaj maxim pentru acea rezolvare, deoarece serviciul de verificare utilizează doar teste simple.

Alte informații

Concurenții ar putea fi descalificați în următoarele situații:

- încearcă să deranjeze un alt concurent în activitatea sa;
- încearcă să întrerupă instalațiile sau facilitățile de evaluare sau
- încearcă să pună în pericol sau să provoace daune concursului.

În scopul de a putea înainta răspunsuri, de a realiza copii și listări, calculatoarele vor fi în rețea internă. Această rețea este construită respectând reguli de securitate.

Concurenților le este interzisă accesarea rețelei în orice alt scop sau cu orice alt instrument decât cele menționate și puse la dispoziția lor de organizatori. Sunt interzise chiar și comenzi simple cum ar fi un `'ping'`. Dacă apare vreun dubiu cu privire la funcționarea rețelei, concurenții trebuie să anunțe supraveghetorul din laborator.

De asemenea, nu li se permite concurenților să producă materiale accesibile rețelei de pe calculatorul lor. Traficul rețelei se va monitoriza pe parcursul concursului, iar concurenții care contravin regulilor, vor putea fi descalificați.

Pagini Web utile

Vă prezentăm o listă de pagini *Web* care ar trebui consultate pentru a participa cu succes la IOI 2001.

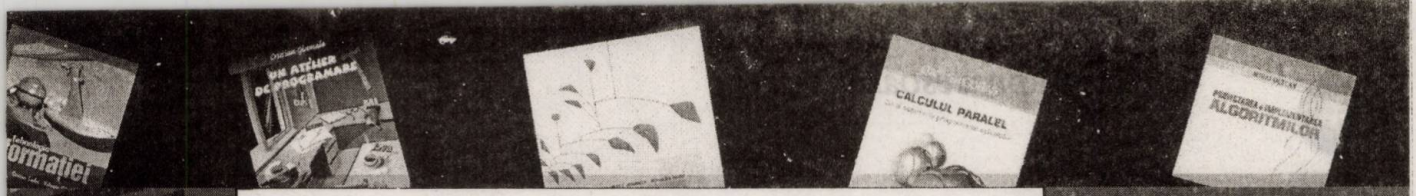
- *Linux Debian release 2.2r2*: **http://www.debian.org;**
- Pachet *DJGPP* pentru cei care nu vor să instaleze *Linux*: **http://www.cygwin.com;**
- *Windows '98 SE*: **http://www.microsoft.com/windows;**
- *Gcc 2.95.2*: **http://gcc.gnu.org;**
- *Freepascal*: **http://www.freepascal.org.**

Pentru o oarecare liniștire a celor care nu se pot obișnui nicicum cu noul, organizatorii vor instala un mediu integrat "*turbo-lookalike*":

http://www.lanet.lv/~pavenis/rhide.html.

Jyrki Nummenmaa
Președintele Comitetului Științific al IOI 2001

Am publicat acest regulament cu toate că, din România, doar patru concurenți vor participa la IOI 2001; poate îi pune pe gânduri pe mulți... într-un fel, este vorba de viitor...



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.

Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

REDUCERI de prețuri

Mihai Scorțaru

Luna aprilie a dus la reduceri semnificative de prețuri atât la procesoarele produse de Intel, cât și la cele produse de AMD. Reducerile nu se vor opri aici; în vara acestui an vor avea loc alte "corecții" ale prețurilor.

Noi produse

La sfârșitul lunii aprilie a fost lansat procesorul *Pentium 4* cu frecvența de 1,7 GHz. Datorită acestei lansări, *Intel* și-a consolidat poziția de lider în lupta cu megahertzii. Totuși, performanțele noului procesor lasă destul de mult de dorit. Noul *Pentium 4* nu este mult mai performant decât predecesorul său care are o frecvență cu 200 MHz mai mică. În ceea ce privește firma *AMD*, aceasta a lansat pe piață noul procesor *Athlon 4*, deocamdată numai în varianta destinată calculatoarelor portabile. Probabil că în momentul în care citiți aceste rânduri, diferitre alte produse noi vor fi disponibile pe piață.

Produse viitoare

Data de 29 mai va reprezenta, probabil, momentul în care procesorul *Intel Itanium* va fi lansat oficial pe piață. În aceeași zi va apărea și varianta pentru servere a procesorului *Pentium 4*, sub numele de *Pentium 4 Xeon*.

Răspunsul celor de la *AMD* nu se lasă prea mult timp așteptat. În scurt timp vom avea variantele pentru servere ale procesorului *Athlon 4*.

Va fi o vară interesantă în industria procesoarelor. Pe piață vor apărea o mulțime de alte produse noi cum ar fi procesoarele *Pentium III* fabricate folosind tehnologia de 0,13 microni, procesoare *Athlon 4* pentru servere și *desktop*-uri, o nouă generație de procesoare *Duron* etc.

Intel

Prețurile procesoarelor *Intel* s-au redus din nou. Și de această dată, cele mai importante reduceri au apărut la familia *Pentium 4*. Prețul procesorului *Pentium III* cu frecvența de 1 GHz a devenit mai mare decât cel al anumitor procesoare *Pentium 4*.

La aceeași frecvență, *AMD* oferă produse mai performante și mai ieftine decât *Intel* atât pentru piața calculatoarelor ieftine, cât și pentru piața calculatoarelor performante.

Celeron

În ultima lună, familia procesoarelor *Celeron* nu s-a îmbogățit cu produse noi. Prețurile și-au păstrat tendința descendentă; reduceri semnificative apar la frecvențe mai mari.

Datorită faptului că pe piață nu există multe procesoare *Celeron 850*, prețul acestui produs nu s-a modificat.

La începutul lunii mai prețurile procesoarelor *Celeron* erau următoarele:

Celeron 566	*46$*	*0$*	*(0%)*
Celeron 600	*50$*	*-3$*	*(-6%)*
Celeron 633	*51$*	*-10$*	*(-16%)*
Celeron 667	*51$*	*+2$*	*(+4%)*
Celeron 700	*60$*	*-8$*	*(-12%)*
Celeron 733	*66$*	*-8$*	*(-11%)*
Celeron 766	*68$*	*-12$*	*(-15%)*
Celeron 800	*83$*	*-17$*	*(-17%)*
Celeron 850	*138$*	*0$*	*(0%)*

Singura creștere de preț a apărut pentru procesorul *Celeron 667*. Luna trecută prețul era de 49$ datorită faptului că se dorea lichidarea anumitor stocuri. Se pare că aceste stocuri s-au epuizat în mare parte, motiv pentru care prețul a atins un nivel relativ normal.

Pentium III

Prețurile procesoarelor *Pentium III* cu magistrala de 133 MHz sunt următoarele:

Pentium III 600	*99$*	*+3$*	*(+3%)*
Pentium III 667	*87$*	*-11$*	*(-11%)*
Pentium III 733	*99$*	*-18$*	*(-15%)*
Pentium III 800	*147$*	*+2$*	*(+1%)*
Pentium III 866	*156$*	*-6$*	*(-4%)*
Pentium III 933	*166$*	*-38$*	*(-19%)*
Pentium III 1000	*199$*	*-25$*	*(-11%)*

Procesorul *Pentium III* la 1 GHz este acum mai scump decât un *Pentium 4* la 1,3 sau 1,4 GHz. Majoritatea prețurilor au scăzut, cu câteva excepții. Se pare că în acest moment frecvența de 733 MHz reprezintă o alegere foarte bună.

Pentium 4

Noul procesor cu frecvența de 1,7 GHz, alături de reducerile generale de prețuri practicate de *Intel*, a dus la scăderea prețurilor produselor din această familie. Reducerile nu se vor opri aici; prețurile vor scădea și mai mult în cursul acestei veri.

La începutul lunii mai procesoarele *Intel Pentium 4* puteau fi achiziționate în Statele Unite la următoarele prețuri:

Pentium 4 1300	*183$*	*-94$*	*(-34%)*
Pentium 4 1400	*186$*	*-113$*	*(-38%)*
Pentium 4 1500	*256$*	*-219$*	*(-46%)*
Pentium 4 1700	*350$*	*n/a*	*n/a*

Procesorul *Pentium 4* cu frecvența de 1,3 GHz nu va mai fi produs de către *Intel*. Acest lucru este indicat de faptul că *Intel* practică, pentru distribuitorii săi, același preț pentru procesoarele cu frecvențe de 1,3 și 1,4 GHz.

Nici actualele procesoare *Pentium III* nu vor mai rezista mult timp pe piață. Tehnologia de 0,18 microni folosită de *Intel* nu permite atingerea de frecvențe mai mari de 1,2 GHz. Trecerea la o tehnologie de 0,13 microni va permite atingerea de frecvențe mai mari, dar este puțin probabil ca arhitectura *Pentium III* (aceeași cu cea de la vechiul *Pentium Pro*) să permită depășirea barierei de 2 GHz. Noua arhitectură *Pentium 4* permite atingerea de viteze de peste 5 GHz (probabil până la 10 GHz), iar trecerea la tehnologia de 0,13 microni va mări și frecvențele procesoarelor din această familie.

Procesorul *Pentium 4* cu frecvența de 2 GHz este așteptat pentru a doua jumătate a acestui an. Bariera de 1 GHz a fost atinsă pe data de 6 martie 2000 de către *AMD* și pe data de 8 martie 2000 de către *Intel*. Se pare că celebra lege potrivit căreia vitezele procesoarelor se dublează la fiecare 18 luni va rămâne în vigoare. Acest lucru este valabil doar pentru *Intel*. Este destul de puțin probabil să apară un procesor *AMD* cu frecvența de 2 GHz în cursul anului 2001.

AMD

Pentru a răspunde la reducerile de prețuri practicate de *Intel*, *AMD* a ieftinit propriile produse. Nu au apărut procesoare *Athlon* sau *Duron* noi, dar pe diferite *site*-uri apar deja oferte de procesoare *Athlon* cu frecvența de 1,4 GHz, deci lansarea oficială nu va întârzia mult.

Duron

Prețurile procesoarelor *Duron* au scăzut și ele. Acestea pot fi achiziționate la următoarele prețuri:

Duron 600	*42$*	*0$*	*(0%)*
Duron 650	*40$*	*-1$*	*(-2%)*
Duron 700	*37$*	*-4$*	*(-10%)*
Duron 750	*35$*	*-6$*	*(-15%)*
Duron 800	*45$*	*-8$*	*(-15%)*
Duron 850	*54$*	*-21$*	*(-28%)*
Duron 900	*69$*	*-23$*	*(-25%)*

Prețurile procesoarelor cu frecvențe de peste 700 MHz au scăzut foarte mult. În acest moment cea mai bună alegere o reprezintă, evident, procesorul *Duron 750*.

Athlon/200

Prețurile procesoarelor *Athlon Thunderbird* cu frecvența magistralei de 200 MHz au scăzut și ele:

Athlon 700	*51$*	*-13$*	*(-20%)*
Athlon 750	*51$*	*-25$*	*(-33%)*
Athlon 800	*80$*	*-11$*	*(-12%)*
Athlon 850	*82$*	*-8$*	*(-9%)*
Athlon 900	*92$*	*-15$*	*(-14%)*
Athlon 950	*122$*	*-14$*	*(-10%)*
Athlon 1000	*123$*	*-14$*	*(-10%)*
Athlon 1100	*137$*	*-12$*	*(-8%)*
Athlon 1200	*140$*	*-47$*	*(-25%)*
Athlon 1300	*168$*	*-48$*	*(-22%)*

Athlon/266

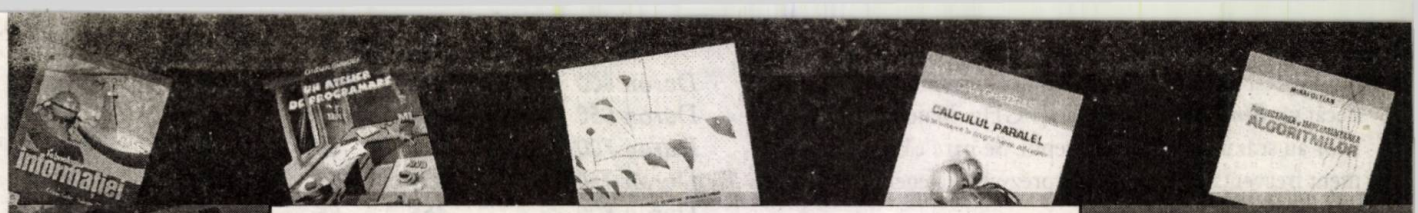
Prețurile procesoarelor *Athlon Thunderbird* cu frecvența magistralei de 266 MHz au scăzut și ele relativ mult. O reducere mai mică apare în cazul procesorului cu frecvența de 1,33 GHz datorită stocurilor limitate. Prețurile minime în Statele Unite sunt următoarele:

Athlon 1000	*129$*	*-19$*	*(-13%)*
Athlon 1133	*147$*	*-28$*	*(-16%)*
Athlon 1200	*148$*	*-44$*	*(-23%)*
Athlon 1333	*198$*	*-19$*	*(-9%)*

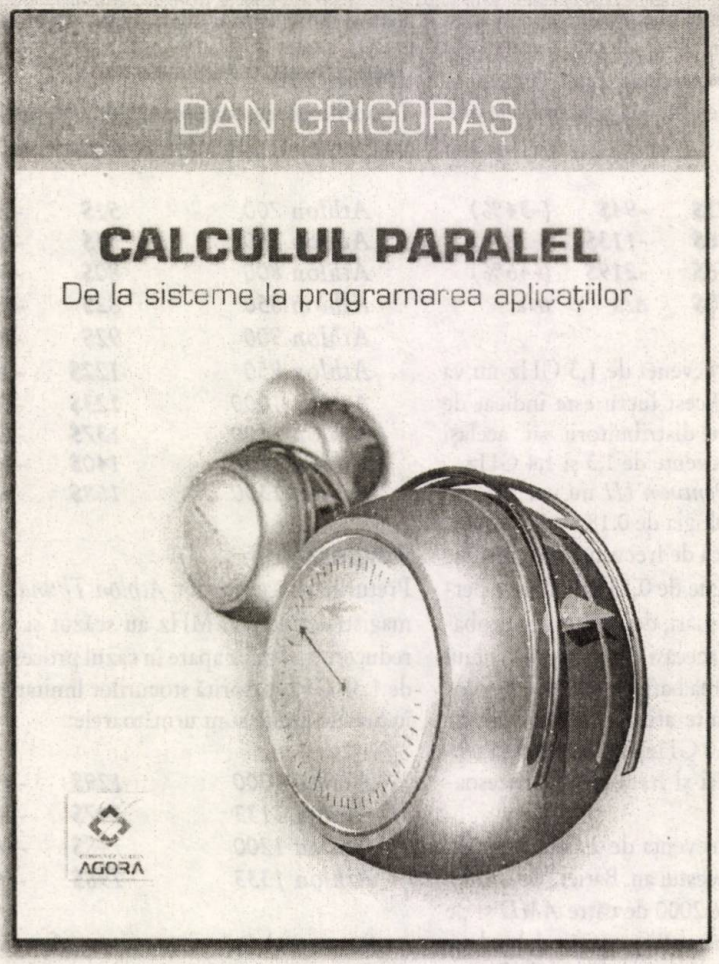
Succesorul acestui procesor a fost lansat pe piață (în versiunea pentru calculatoare portabile) pe data de 14 mai 2001. Până acum acest succesor era cunoscut doar sub numele de cod *Palomino*. Denumirea oficială a produsului este *Athlon 4*. Nu au existat și nu vor exista procesoare cu denumirea *Athlon 2* sau *Athlon 3*. Numele a fost ales pentru a indica faptul că acest procesor va fi rivalul procesorului *Intel Pentium 4*. Procesorul *AMD Mobile Athlon 4* este disponibil în variante cu frecvențe de 850, 900, 950 și 1000 MHz.

De acum pe Web

S-a încheiat un an în care am comentat, în fiecare lună, situația din războiul procesoarelor. Începând cu toamna acestui an, pe noul *site* ***GInfo***, vom avea o rubrică în cadrul căreia vom trata acest subiect. Așteptăm sugestii pe adresa redacției.



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate, dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Sisteme de FIȘIERE

Ciprian Ileană, Claudiu Soroiu

În cadrul acestui articol vom continua prezentarea sistemului de fișiere folosit de sistemele de operare produse de Microsoft. În articolul din numărul anterior am prezentat câteva structuri de date importante prezente pe un volum FAT. Vom descrie acum modul în care pot fi inițializate volumele FAT, structura unor sectoare specifice sistemului FAT32, structura directoarelor, intrările cu nume lungi din FAT etc.

Inițializarea volumelor FAT

Sistemele de operare *Microsoft* folosesc doar sistemul *FAT12* pentru dischete. Deoarece numărul formatelor de dischete care au aceeași dimesiune este limitat, inițializarea *BPB*-ului se realizează cu ajutorul unui tabel simplu.

Nu există nici o metodă dinamică de calcul pentru *FAT12*. Astfel, pentru formatele *FAT12* toate calculele pentru `BPB_SecPerClus` și `BPB_FATSz16` au fost realizate folosindu-se foaia și creionul, iar rezultatele au fost înregistrate în tabele. Una dintre condițiile impuse a fost, evident, ca numărul de *cluster*-e rezultat să fie mai mic decât 4085. În cazul în care discul (partiția) folosit(ă) are o capacitate mai mare de 4 MB folosirea sistemului *FAT12*, nu mai este recomandată. Este indicată folosirea unei valori mai mici pentru `BPB_SecPerClus`, astfel încât volumul să devină *FAT16*.

Din acest punct și până la sfârșitul acestei secțiuni ne vom referi doar la discuri (partiții) care au 512 bytes pe sector. Trebuie avut în vedere faptul că informațiile din această secțiune nu le veți putea folosi decât în cazul discurilor (partițiilor) care au 512 bytes pe sector.

Există o valoare fixată care face separarea între *FAT16* și *FAT32*. Orice volum cu o dimensiune mai mică decât aceasta este *FAT16* și orice volum de această dimensiune, sau mai mare, este *FAT32*. De exemplu, pentru *Windows* această valoare este de 512 MB. Orice volum *FAT* mai mic de 512 MB este *FAT16* și orice volum *FAT* de 512 MB, sau mai mare, este *FAT32*. Totuși, există multe volume *FAT16* care sunt mai mari de 512 MB, datorită variatelor modalități prin care se poate forța utilizarea unui *FAT16* în locul prestabilitului *FAT32* prin modificarea limitelor în care acestea se încadrează.

Acum vom vorbi despre limita prestabilită pentru volumele *MS-DOS* și *Windows* care nu au fost încă formatate. Există două tabele, unul pentru *FAT16* iar celălalt pentru *FAT32*.

```
struct DskSzToSecPerClus{
  DWORD DiskSize;
  BYTE SecPerClusVal;
};

/* Tabela pentru volume FAT16. */
DskSzToSecPerClus DskTableFAT16[]={
  {8400,0}, // discuri cu capacitate de
            // pana la 4.1 MB, valoarea
            // 0 pentru SecPerClusVal
            // reprezinta o eroare
  {32680,2}, // discuri cu capacitate de
             // pana la 16 MB,cluster de
             // 1k
  {262144,4}, // discuri cu capacitate
              // de pana la 128 MB,
              // cluster de 2k
  {524288,8}, // discuri cu capacitate
              // de pana la 256 MB,
              // cluster de 4k
  {1048576,16}, // discuri cu capacitate
                // de pana la 512 MB,
                // cluster de 8k
// Intrarile care urmeaza nu sunt folosite
// decat in cazul in care FAT16 este fortat
  {2097152,32}, // discuri cu capacitate
                // de pana la 1 GB,
                // cluster de 16k
  {4194304,64}, // discuri cu capacitate
                // de pana la 2 GB,
                // cluster de 32k
```

```
  {0xFFFFFFFF,0} // orice disc cu o
                 // capacitate mai mare
                 // de 2GB, valoarea 0
                 // pentru SecPerClusVal
                 // reprezinta o eroare
};

/* Tabela pentru volumele FAT32. */
DSKSzToSecPerClus DskTableFAT32[]={
  {66600,0}, // discuri cu capacitate de
             // pana la 32.5 MB,
             // valoarea 0 pentru
             // SecPerClusVal reprezinta
             // o eroare
  {532480,1}, // discuri cu capacitate
              // de pana la 260 MB,
              // cluster de 5k
  {16777216,8}, // discuri cu capacitate
                // de pana la 8 GB,
                // cluster de 4k
  {33554432,16}, // discuri cu capacitate
                 // de pana la 16 GB,
                 // cluster de 8k */
  {67108864,32}, // discuri cu capacitate
                 // de pana la 32 GB,
                 // cluster de 16k
  {0xFFFFFFFF,64} // discuri cu capaci-
                  // tate mai mare de
                  // 32GB, cluster de 32k
};
```

Așadar, fiind dată dimensiunea unui disc și un tip de *FAT* (*FAT16* sau *FAT32*), avem acum o valoare pentru `BPB_SecPerCluster`. Singura operație care mai trebuie efectuată este calcularea numărului de sectoare ocupate de structura *FAT* pentru a putea seta valoarea `BPB_FATSz16` sau `BPB_FATSz32`.

În cele ce urmează vom presupune că `BPB_RootEntCnt`, `BPB_RsvdSecCnt` și `BPB_NumFATs` sunt setate corect. De asemenea, presupunem că variabila `DskSize` reprezintă mărimea volumului pe care o vom atribui variabilei `BPB_TotSec32` sau variabilei `BPB_TotSec16`.

```
RootDirSectors=((BPB_RootEntCnt*32)+
      (BPB_BytsPerSec-1))/BPB_BytsPerSec;
TmpVal1=DskSize-(BPB_ResvdSecCnt+
                      RootDirSectors);
TmpVal2=(256*BPB_SecPerClus)+BPB_NumFATs;
if (FATType==FAT32)
  TmpVal2=TmpVal2/2;
FATSz=(TMPVal1+(TmpVal2-1))/TmpVal2;
if (FATType==FAT32){
  BPB_FATSz16=0;
  BPB_FATSz32=FATSz;
}
else
  BPB_FATSz16 = LOWORD(FATSz);
  // nu exista BPB_FATSz32 intr-un BPB de
  // FAT16
```

Matematica ce stă la baza calculelor din codul de mai sus este foarte complicată. Important este faptul că sistemele de operare *Microsoft* funcționează folosindu-se de aceste calcule. Totuși, aceste calcule matematice nu funcționează perfect. Pot apărea erori cum ar fi setarea valorii

Structura sectorului FSInfo

Tabelul 1

Denumire	*Offset*	*Dimensiune*	*Descriere*
`FSI_LeadSig`	0	4 bytes	Are valoarea `0x41615252`. Această semnătură este folosită pentru a valida faptul că acest sector este un sector `FSInfo`.
`FSI_Reserved1`	4	480 bytes	Acest câmp este în prezent rezervat pentru dezvoltări ulterioare; codul *FAT32* ar trebui să inițializeze acești bytes cu `0` și nu ar trebui să fie folosiți în prezent.
`FSI_StrucSig`	484	4 bytes	Are valoarea `0x61417272`; reprezintă o altă semnătură care este adesea folosită în sector, în apropierea câmpurilor folosite.
`FSI_Free_Count`	488	4 bytes	Conține ultimul *cluster* liber cunoscut; dacă valoarea sa este `0xFFFFFFFF` înseamnă că ultimul *cluster* liber nu este cunoscut și trebuie determinat folosind alte metode.
`FSI_Reserved2`	496	12 bytes	Acest câmp este în prezent rezervat pentru dezvoltări ulterioare; codul *FAT32* ar trebui să inițializeze acești bytes cu `0` și nu ar trebui să fie folosiți în prezent.
`FSI_TrailSig`	508	4 bytes	Are valoarea `0xAA550000`; această semnătură este folosită pentru a valida faptul că acest sector este un sector `FSInfo`. Trebuie remarcat faptul că valorile octeților cu *offset*-urile 510 și 511 sunt identice cu valorile octeților corespunzători din sectorul 0.

FATSz cu până la două sectoare în plus în cazul unui *FAT16* și cu până la opt sectoare în plus în cazul unui *FAT32*.

Dacă în urma calculelor valoarea rezultată pentru FATSz va fi mai mică decât cea reală, aceasta va determina funcționarea defectoasă a *FAT*-ului. În concluzie, este preferată metoda de calcul de mai sus, cu toate că în unele cazuri se vor irosi câteva sectoare (este cunoscut faptul că *FAT*-ul funcționează corect și în cazul unei valori mai mari decât cea reală pentru FATSz).

Sectoarele FSInfo și BkBoot

În cazul unui volum *FAT32*, structura *FAT* nu are o limită de mărime propriu-zisă așa cum este în cazul *FAT16* și *FAT12*. Din acest motiv, este alocat un spațiu destinat stocării ultimului *cluster* liber cunoscut din cadrul volumului *FAT32*, optimizând astfel lucrul aplicațiilor care în loc să calculeze mereu valoarea acestui *cluster* vor avea acum acces direct la aceasta. Numărul sectorului FSInfo se regăsește în valoarea câmpului BPB_FSInfo; în cazul sistemelor de operare *Microsoft* această valoare este setată întotdeauna cu 1. Structura sectorului FSInfo este prezentată în tabelul 1.

O altă facilitate existentă în *FAT32*, dar nu și în *FAT16* sau *FAT12*, este prezența câmpului BPB_BkBootSec. Cu toții știm că volumele *FAT16/FAT12* pot fi în totalitate pierdute în cazul în care conținutul sectorului 0 este rescris sau este distrus nemaiputând fi citit. Prezența câmpului BPB_BkBootSec (are valoarea 6 întotdeauna) în cazul volumelor *FAT32* reduce șansele de pierdere a informațiilor conținute în respectivul volum, deoarece pe al BPB_BkBootSec-ulea sector se află o copie la zi a sectorului de *boot*, copie ce conține și *BPB*-ul volumului.

În cazul în care informațiile din sectorul 0 au fost rescrise din greșeală, utilitarele de disc au simpla sarcină de a restaura datele originale folosindu-se de copia prezentă în sectorul de *back-up*. Dacă sectorul 0 devine inutilizabil și prin urmare nu mai poate fi citit, prezența sectorului de *back-up* permite folosirea volumului în continuare, însă este recomandată salvarea informației și schimbarea discului. Acest al doilea caz (sectorul 0 devine "bad") este motivul pentru care câmpul BPB_BkBootSec nu ar trebui să conțină altă valoare decât 6. Mai exact, multe din sistemele de operare caută acest sector de *back-up* începând cu al șaselea sector al volumului *FAT32*.

Acest așa-zis sector de *back-up* ocupă, în realitate, trei sectoare, fiecare având o lungime de 512 bytes. De asemenea, trebuie remarcat faptul că, la fel ca și sectorul de *boot*, cele trei sectoare conțin la *offset*-urile 510 și 511 semnătura 0xAA55.

Structura directoarelor FAT

Pentru început vom ignora intrările de directoare cu nume lungi și ne vom ocupa doar de intrările de directoare cu nume scurte. Un director din *FAT* nu este altceva decât un "fișier" compus dintr-o listă liniară de structuri pe 32 de bytes. În cazul volumelor *FAT12* și *FAT16* directorul rădăcină are o locație fixă pe disc (imediat după ultimul *FAT*) și o dimensiune fixă calculată în sectoare (vezi calculele pentru RootDirSectors din numărul anterior al revistei *GInfo*); în acest caz primul sector al directorului rădăcină este relativ la primul sector al volumului *FAT*:

```
FirstRootDirSecNum BPB_ResvdSecCnt+
    (BPB_NumFATs*BPB_FATSz16);
```

În cazul sistemului *FAT32*, directorul rădăcină poate avea dimensiune variabilă și este o listă de *cluster*-e, la fel ca orice alt director. Primul *cluster* al directorului rădăcină pe volumele *FAT32* este stocat în BPB_RootClus. Spre deosebire de celelalte directoare, directorul rădăcină nu are setate o dată și o oră, nu poate avea alt nume în afară de "\" și nu conține fișierele "." și ".." ca fiind primele două intrări în director. O mare deosebire față de directoarele obișnuite este aceea că directorul rădăcină este singurul director din cadrul volumului *FAT* care poate avea setat doar bitul corespunzător atributului ATTR_VOLUME_ID.

Structura *FAT* a unui director este prezentată în tabelul 2.

DIR_Name

Dacă DIR_Name[0]=0xE5, atunci intrarea directorului este liberă (nu există nici un nume de fișier sau de director în această intrare).

Dacă DIR_Name[0]=0x00, atunci intrarea directorului este liberă (la fel ca în cazul 0xE5) și după această intrare în director nu mai este alocată nici o altă intrare de director (toți octeții DIR_Name[0], din toate intrările aflate după aceasta, sunt setați cu valoarea 0). Valoarea 0, spre deosebire de 0xE5, indică *driver*-ului sistemului *FAT*, faptul că celelalte intrări de directoare nu mai trebuie examinate deoarece se știe deja că sunt libere.

Dacă Dir_Name[0]=0x05, atunci acest byte are, de fapt, valoarea 0xE5. Valoarea 0x05 este folosită pentru a evita confundarea caracterului cu marcajul unei intrări libere de director.

Valorile nefolosite pentru numele directorului sunt completate cu spații (caractere cu codul 0x20). Totuși, DIR_Name[0] nu poate avea valoarea 0x20.

Între numele și extensia unui director apare caracterul implicit '.'. Acest caracter nu este stocat în DIR_name.

Câmpul DIR_name nu poate conține nici unul dintre caracterele:

- literele mici ale alfabetului;
- valori cuprinse între 0x01 și 0x19, cu excepția valorii 0x05;
- valorile 0x22, 0x2A, 0x2B, 0x2C, 0x2E, 0x2F, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x5B 0x5C, 0x5D și 0x7C.

În *FAT* numele tuturor directoarelor sunt unice.

DIR_Attr

Câmpul DIR_Attr este folosit pentru a specifica atributele unui fișier.

Semnificațiile biților sunt următoarele:

Intrarea pentru un director

Tabelul 2

Denumire	*Offset*	*Dimensiune*	*Descriere*
DIR_Name	0	11 bytes	Nume scurt.
DIR_Attr	11	1 byte	Atributele fișierului: ATTR_READ_ONLY - bitul 0 ATTR_HIDDEN - bitul 1 ATTR_SYSTEM - bitul 2 ATTR_VOLUM_ID - bitul 3 ATTR_DIRECTORY - bitul 4 ATTR_ARCHIVE - bitul 5 Cei mai semnificativi doi biți din byte-ul atributelor sunt rezervați și ar trebui întotdeauna setați pe 0 la crearea fișierului fără a fi ulterior modificați.
DIR_NTRes	12	1 byte	Rezervat pentru a fi folosit de *Windows NT*; valoarea este setată cu 0 la crearea fișierului fără a fi ulterior modificată.
DIR_CrtTimeTenth	13	1 byte	Milisecundele din momentul creării fișierului. În realitate, sunt numărate doar zecimile de secundă.
DIR_CrtTime	14	2 bytes	Ora la care a fost creat fișierul.
DIR_CrtDate	16	2 bytes	Data la care a fost creat fișierul.
DIR_LstAccDate	18	2 bytes	Data ultimei accesări (citire/scriere).
DIR_FstClusHI	20	2 bytes	Cei mai semnificativi 16 biți ai numărului primului *cluster* al intrării (pentru *FAT12* sau *FAT16* are întotdeauna valoarea 0).
DIR_WrtTime	22	2 bytes	Ora ultimei modificări (creare/scriere).
DIR_WrtDate	24	2 bytes	Data ultimei modificări (creare/scriere).
DIR_FstClusLO	26	2 bytes	Cei mai semnificativi 16 biți ai numărului primului *cluster* al intrării.
DIR_FileSize	28	4 bytes	Dimensiunea fișierului pe 32 de biți (DWORD), exprimată în bytes.

- ATTR_READ_ONLY - indică faptul că scrierea asupra fișierului ar trebui să eșueze.
- ATTR_HIDDEN - indică faptul că la o listare obișnuită a fișierelor, acesta nu ar trebui afișat.
- ATTR_SYSTEM - indică faptul că acesta este un fișier al sistemului de operare.
- ATTR_VOLUME_ID - ar trebui să existe un singur astfel de "fișier" pe fiecare volum, iar acesta ar trebui să fie directorul rădăcină; numele acestui fișier este, de fapt, eticheta volumului. DIR_FstClusHI și DIR_FstClusLO trebuie să aibă întotdeauna valoarea 0 pentru etichetele de volum (pentru fișierele de etichetă nu se alocă nici un *cluster* de date).
- ATTR_DIRECTORY - indică faptul că acest fișier este un director (*container* de fișiere) care conține intrările altor fișiere.
- ATTR_ARCHIVE - acest atribut vine în ajutorul utilitarelor de *back-up*; *driver*-ul sistemului de fișiere *FAT* setează acest bit în cazul în care fișierul a fost creat, redenumit sau modificat; astfel utilitarele de *back-up* pot folosi acest atribut pentru a indica fișierele din volum care au fost modificate de la ultimul *back-up* efectuat.

În cazul în care atributul ATTR_LONG_NAME este setat acesta indică faptul că "fișierul" este parte a unui nume lung al unui alt fișier.

Când un director este creat (un fișier care are setat atributul ATTR_DIRECTORY din câmpul DIR_Attr), câmpul DIR_FileSize este setat cu 0 deoarece directoarele sunt dimensionate prin simpla urmărire a listei de *cluster*-e până la întâlnirea marcajului *EOC*. La creare, directorului i se alocă un *cluster* (acest lucru nu este valabil și în cazul sistemelor *FAT16/FAT12*) al cărui număr va fi scris în DIR_FstClusLO și DIR_FstClusHi; după aceasta la intrarea acestui *cluster* în *FAT* se va scrie marcajul *EOC*. În cazul în care directorul creat este directorul rădăcină, nu mai trebuie efectuate alte operații.

Dacă se creează un director obișnuit, vor trebui create două intrări speciale în primii 32 de bytes ai intrării directorului (primii 32 de bytes din zona de date a *cluster*-ului care tocmai a fost alocat).

Aceste două intrări vor avea valoarea DIR_Name setată astfel:

- pentru prima avem DIR_Name = ".   ".
- pentru a doua avem DIR_Name = "..   ".

Aceste două intrări poartă numele de intrări *dot* (punct) și *dotdot* (punct punct). Ceea ce este important de reținut la aceste intrări este faptul că intrarea *dot* este un director care se referă la el însuși, iar intrarea *dotdot* referă *cluster*-ul directorului părinte (acesta este 0 în cazul în care directorul părinte este directorul rădăcină).

Formatele de dată și timp

Multe sisteme de fișiere *FAT* nu suportă decât câmpurile `DIR_WrtTime` și `DIR_WrtDate`. Din acest motiv, câmpurile `DIR_CrtTimeMil`, `DIR_CrtTime`, `DIR_CrtDate` și `DIR_LstAccDate` sunt opționale (este obligatorie tratarea câmpurilor `DIR_WrtTime` și `DIR_WrtDate`). În cazul în care aceste câmpuri nu sunt tratate, acestea ar trebui să fie inițializate cu valoarea 0 și ignorate de către celelalte operații cu fișiere.

Data este stocată într-o intrare *FAT* pe16 biți și este relativă la "*era MS-DOS*" (prima dată este 01/01/1980). Formatul datei este prezentat mai jos (bitul 0 este *LSB* iar bitul 15 este *MSB*):

- biții `0-4`: ziua lunii, cu valori valide între `1` și `31` (inclusiv).
- biții `5-8`: luna anului, cu valori valide între `1` și `12` (inclusiv).
- biții `9-15`: numărul de ani scurși din `1980` și până în prezent, cu valori valide între `0` și `127` (inclusiv); se acoperă perioada `1980-2107`.

Timpul este stocat într-o intrare *FAT* pe 16 biți cu o granulație de două secunde. Formatul timpului este prezentat mai jos (bitul 0 este *LSB* iar bitul 15 este *MSB*):

- biții `0-4`: interval de două secunde, cu valori valide între `0` și `29` (inclusiv); se acoperă intervalul `0-58` secunde).
- biții `5-10`: minutele, cu valori valide între `0` și `59` (inclusiv).
- biții `11-15`: ora, cu valori valide între `0` și `23` (inclusiv).

Se acoperă astfel un interval cuprins între `00:00:00` și `23:59:58` (acesta este intervalul valorilor valide ale timpului).

Intrări cu nume lungi

Adăugarea intrărilor cu nume lung în *FAT* a venit într-un moment crucial pentru design-ul existent al *FAT*-ului:

- Modificările trebuiau să fie transparente pentru versiunile anterioare ale sistemului de operare *MS-DOS*. Scopul principal este acela, ca aplicațiile realizate pentru versiunile anterioare de *MS-DOS/Windows* să nu poată accesa cu ușurință intrările cu nume lung. Singurele aplicații *MS-DOS* care pot "să vadă" intrările cu nume lung sunt aplicațiile bazate pe căutare utilizând *FCB*-ul (*File Control Block*), acest lucru se întâmplă numai în cazul în care folosesc masca completă de căutare cu caractere de control (`*.*`) și atributul complet de căutare cu toți biții setați (`0xFF`). În versiunile de *MS-DOS/Windows* anterioare sistemului de operare *Windows '95*, nici o aplicație *MS-DOS* nu putea "să găsească" nici măcar o singură intrare cu nume lung.
- O intrare cu nume lung trebuie să fie alocată fizic, pe disc, în apropierea intrării cu nume scurt la care aceasta face referire. După cum se va vedea mai târziu, intrările cu nume lung se află imediat înaintea intrării cu nume

Intrările cu nume lung — Tabelul 3

Denumire	*Offset*	*Dimensiune*	*Descriere*
`LDIR_Ord`	0	1 byte	Numărul de ordine al intrării în setul de intrări pentru numele lung atașat unui nume scurt.
`LDIR_Name1`	1	10 bytes	Primele cinci caractere din subcomponenta numelui lung atașat unei intrări de director; se folosește setul de caractere *UNICODE*.
`LDIR_Attr`	11	1 byte	Atributele intrării - bitul `ATTR_LONG_NAME` trebuie să aibă valoarea 1.
`LDIR_Type`	12	1 byte	Dacă are valoarea 0, atunci intrarea este o sub-componentă a unui nume lung. Alte valori sunt rezervate pentru versiunile următoare. O valoare diferită de 0 indică o intrare de alt tip.
`LDIR_Chksum`	13	1 byte	Suma de control a numelui din intrarea de director cu nume scurt.
`LDIR_Name2`	14	12 bytes	Caracterele `6-11` din subcomponenta numelui lung din această intrare a directorului.
`LDIR_FstClusLO`	26	2 bytes	Trebuie să aibă valoarea 0 pentru compatibilitatea cu versiunile precedente ale sistemelor de operare *Microsoft*.
`LDIR_Name2`	28	4 bytes	Ultimele două caractere din subcomponenta numelui lung din această intrare a directorului.

scurt cu care sunt asociate, și existența lor are un impact insesizabil asupra performanței sistemului de fișiere.

- Dacă un utilitar de disc detectează intrări cu nume lung, acesta nu pune în pericol integritatea datelor existente în fișiere. Utilitarele de disc, de obicei, nu utilizează aplicațiile *MS-DOS* ca să acceseze structurile de date specifice sistemului de fișiere de pe disc. Acestea citesc informația din sectorul fizic sau logic de pe disc și verifică ce conține intrarea din director. Intrările cu nume lung au fost adăugate sistemului de fișiere *FAT* astfel încât să nu cauzeze pierderi de date pe un disc care conține aceste tip de intrări dacă discul a fost "reparat" de un utilitar compatibil doar cu versiunile de *MS-DOS/Windows* anterioare sistemului de operare *Windows '95*.

Din dorința celor de la *Microsoft*, ca numele lungi să se impună pe aceleași structuri *FAT* care au fost construite începând cu prima versiune *MS-DOS*, intrările cu nume lung sunt definite ca o intrare cu nume scurt, dar cu atribute speciale. O intrare cu nume lung este o intrare obișnuită cu nume scurt în care câmpul atribut are valoarea:

```
ATTR_LONG_NAME=ATTR_READ_ONLY|
ATTR_HIDDEN|ATTR_SYSTEM|ATTR_VOLUME_ID;
```

Pentru a determina dacă o intrare este o componentă a unui nume lung, masca atributului este următoarea:

```
ATTR_LONG_NAME=ATTR_READ_ONLY|
ATTR_HIDDEN|ATTR_SYSTEM|ATTR_VOLUME_ID|
ATTR_DIRECTORY|ATTR_ARCHIVE;
```

Când este întâlnită o astfel de intrare, aceasta este tratată separat de sistemul de fișiere (operare). Aceste intrări sunt tratate ca o parte a unui set de intrări din director care sunt asociate unei singure intrări cu nume scurt din director. Fiecare intrare de nume lung are structura din tabelul 3.

Organizarea și asocierea dintre intrările cu nume lung și cele cu nume scurt

Fiecărei intrări cu nume scurt îi precede un set de intrări cu nume lung. Intrările cu nume lung sunt asociate intrării cu nume scurt dintr-un motiv foarte simplu: numai intrările cu nume scurt sunt vizibile sistemelor de operare anterioare sistemului *Windows '95*. Fără intrarea cu nume scurt asociată, intrările cu nume lung sunt complet invizibile acestor sisteme de operare. O intrare cu nume lung nu poate fi interpretată dacă nu se găsește intrarea cu nume scurt căreia îi este asociată. Dacă există intrări cu nume lung cărora nu le este asociată nici o intrare validă cu nume scurt, aceste intrări poartă numele de "orfani".

În primul rând, fiecare componentă a unui set de intrări cu nume lung este numerotat în mod unic, și valoarea numărului, ultimei componente a setului este calculată prin operația OR (SAU logic) aplicată între acesta și un număr care indică faptul că acest component este ultimul din set. Câmpul `LDIR_Ord` este folosit pentru a realiza această determinare a ultimului component al setului. Prima componentă a setului are valoarea `1` și este păstrată în câmpul `LDIR_Ord`. A *n*-a componentă are valoarea `n OR LAST_LONG_ENTRY` pentru acest câmp. Acest câmp (`LDIR_Ord`) nu poate avea nici una dintre valorile `0x00` sau `0xE5`. Aceste două valori au fost utilizate de la început de sistemul de fișiere pentru a indica o intrare "liberă" în director, respectiv "ultima" intrare de director din *cluster*. Câmpul `LDIR_Ord` nu poate avea aceste valori, doarece nu se află în domeniul variabilei (valorile pe care le poate avea variabila `LDIR_Ord` sunt cuprinse între 1 și `n OR LAST_LONG_ENTRY`). În cazul în care `LDIR_Ord` depășește domeniul se consideră că intrarea este alterată și este tratată ca un *orfan* de către sistemul de fișiere.

În al doilea rând, se calculează o sumă de control pe 8 biți, pe baza numelui scurt din intrare cu nume scurt din director, în momentul în care sunt create intrările cu nume lung și scurt. Pentru a calcula această sumă de control se folosesc toate cele 11 caractere ale numelui din intrarea cu nume scurt. Suma de control este stocată în fiecare intrare cu nume lung atașată intrării cu nume scurt. Dacă una dintre sumele de control din setul de intrări cu nume lung nu corespunde cu suma de control a numelui conținut de intrarea cu nume scurt, atunci intrările cu nume lung sunt tratate ca orfani. Acest lucru se întâmplă atunci când intrările cu nume scurt sunt citite de o versiune mai veche de *MS-Dos/Windows* și la o redenumire a unei intrări care are atașat numele lung, se schimbă doar numele scurt.

Algoritmul de calculare a sumei de control este:

```
unsigned char ChkSum (unsigned char
                                *pFcbName){
  short FcbNameLen;
  unsigned char Sum;
  Sum =0;
  for (FcbNameLen=11;FcbNameLen;
                                FcbNameLen--)
    Sum=((Sum&1)?0x80:0)+(Sum>>1)+
                                *pFcbName++;
  return Sum;
}
```

Intrarea cu nume scurt este o structură care conține câmpuri cum ar fi data ultimei accesări, ora creării, data creării, primul *cluster*, mărime etc. De asemenea, intrarea cu nume scurt conține un nume vizibil pentru versiunile mai vechi de *MS-DOS/Windows*. Intrările cu nume lung stochează informații noi și nu este nevoie să conțină informații deja existente în intrarea cu nume scurt. De fapt, intrările cu nume lung conțin numele lung al unui fișier. Numele conținut de intrarea cu nume scurt, căreia îi este asociat un set de intrări cu nume lung, poartă denumirea de *alias al fișierului*.

Fișiere și subdirectoare cu nume lung

Un nume lung poate conține mai multe caractere decât încap într-o singură intrare din director. Când acest lucru se întâmplă, numele lung este stocat în mai multe intrări cu nume lung. În orice caz, câmpurile ce conțin numele, din interiorul unei intrări cu nume lung, sunt separate. Următorul exemplu ilustrează cum un nume lung este stocat în mai multe intrări din director. Numele lungi se termină cu

caracterul NULL (0x00) și completate cu 0xFFFF, pentru a evita coruperea câmpurilor unei intrări cu nume lung, de către utilitarele de disc existente deja. Dacă lungimea unui nume lung se potrivește exact în *n* intrări cu nume lung din director (un întreg multiplu de 13), atunci acesta nu se mai termină cu caracterul NULL și nu mai necesită completarea cu 0xFFFF.

Să presupunem că pe disc există un fișier cu numele: The quick brown.fox. Următorul exemplu ilustrează cum este împachetat numele în intrări cu nume lung și scurt. Majoritatea câmpurilor din intrările în directoare sunt și ele prezente.

42h	w	n	.	f	o	0Fh	00h	chk-sum	x		
00h	FFFFh	FFFFh	FFFFh	FFFFh	0000h	FFFFh	FFFFh				
01h	T	h	e	q	0Fh	00h	chk-sum	u			
i	c	k	b	0000h	r	o					
T	H	E	Q	U	I	~	1	F	O	X	20h
NT	Rsvd	Created Time									
Created Date	Last Access Date	0000h	Last Modified Time	Last Modified Date	First Cluster	File Size					

Limita numelor și seturile de caractere

În această secțiune vă vom prezenta restricțiile care apar la numele fișierelor cu privire la numărul de caractere și seturile de caractere admise.

Intrări cu nume scurt

Numele scurte sunt limitate la 8 caractere urmate de un "." opțional și de o extensie de până la 3 caractere lungime. Lungimea totală a căii către fișier nu poate fi mai mare de 80 de caractere (64 de caractere pentru cale, 3 caractere pentru litera discului, 12 caractere pentru nume, punct și extensie și caracterul NULL). Caracterele pot fi litere, cifre sau caractere care au codul mai mare decât 127. Mai pot fi folosite și următoarele caractere speciale:

$ % ' - _ @ ~ ` ! ( ) { } ^ # &.

Numele sunt stocate în pagina de cod *OEM* (*Original Equipment Manufacturer*) pentru care sistemul este configurat în momentul în care este creată o nouă intrare în director. Intrările cu nume scurt se pastrează în formatul *OEM*, pentru compatibilitate cu versiuni anterioare de *MS-DOS/Windows*. Caracterele *OEM* sunt caractere pe 8 biți sau pot fi bazate pe perechi de caractere *DBCS* (*Double-Byte Character Set*) pentru anumite pagini de cod.

Numele scurte sunt acceptate de către sistemul de fișiere dacă se folosesc majusculele. O problemă care apare pentru aproape toate paginile de cod *OEM* este aceea că ele dețin informații despre conversia literelor mici în litere mari corespunzătoare, pentru caracterele extinse, dar o literă mare nu corespunde unei singure litere mici. Acest lucru se întâmplă din cauză că mai multe caractere extinse pot avea aceeași majusculă. Acest lucru creează probleme, deoarece nu se păstrează informația pe care o exprimă un singur caracter extins. Această legătură dintre literele mari și cele mici, are și câteva avantaje printre care acela că previne crearea unor fișiere cu nume care ar părea diferite, dar din cauza acestei legături, prin convertirea la litere mari se obține același nume de fișier.

Intrările cu nume lung

Numele lungi sunt limitate la 255 de caractere, fără caracterul NULL de la sfârșit. Lungimea totală a unui nume lung nu poate depăși 260 de caractere, inclusiv caracterul NULL. Caracterele admise sunt aceleași ca și în cazul numelor scurte, cu precizarea că "." poate fi folosit de mai multe ori într-un nume lung. Următoarele șase caractere pot fi folosite într-un nume lung, în plus față de cele permise la numele scurte: + , ; = [ ].

Spațiile dintr-un nume lung care se află la începutul și sfârșitul numelui, sunt ignorate. Sunt ignorate și caracterele "." care se află la sfârșitul numelui.

Numele lungi sunt stocate în intrările cu nume lung în format *UNICODE*. Nu este posibilă stocarea numelui scurt în format *UNICODE*, deoarece în acest caz caracterele sunt reprezentate pe 8 biți sau caractere *DBCS*.

Când sistemul de fișiere citește numele lung, nu mai face conversia la majuscule, deci numele lung se găsește în forma originală. *UNICODE*-ul rezolvă problema de conversie între literele mari și mici care apare în unele pagini de cod *OEM*, prin conversia la o singură literă mică, și invers, conversia la o majusculă unică.

Legătura dintre numele scurt și cel lung

Numele conținute în setul de intrări cu nume scurt în director poartă denumirea de ***spațiul numelui scurt***, iar numele conținute in setul de intrări cu nume lung poartă denumirea de ***spațiul numelui lung***. Împreună, ele formează un singur spațiu unificat în care nu există duplicate. Aceasta înseamnă că orice nume de fișier/director, fie el lung sau scurt, este unic (apare o singură dată în lista de fișiere și directoare). Mai mult, chiar dacă pentru numele lungi de fișiere/directoare este permisă și utilizarea literelor mici, nu pot exista două fișiere/directoare care să aibă același nume (după conversie). De exemplu, dacă dorim să creăm numele director, acesta nu poate fi creat dacă există deja pe disc, în directorul de lucru un fișier cu numele DIRECTOR sau cu numele lung Director.

Toate tipurile de operații ale sistemului de fișiere (căutare, deschidere, creare, ștergere, redenumire) sunt insensibile la literele mari și mici, deci dacă dorim să deschidem fișierul DIRECTOR, sistemul de fișiere va deschide automat fișierul care există, fie cu numele director sau Director. Aceleași reguli se aplică și în cazul caracterelor extinse.

O căutare pentru nume scurte de fișiere, verifică numai numele scurte dintr-un director, în timp ce o căutare pentru nume lungi de fișiere verifică și numele scurte și pe cele lungi. În timp ce sistemul de operare parcurge directorul, el determină numele lungi de fișiere conținute de sistemul de fișiere. Când se întâlnește un nume scurt, care este asociat numelui lung, operația de căutare pentru nume

lungi verifică mai întâi dacă numele lung și după aceea cel scurt se potrivesc cu masca de căutare.

Dacă un caracter de pe disc (stocat în format *OEM* sau *UNICODE*) nu poate fi tradus într-un caracter corespunzător paginii de cod folosite (*OEM* sau *ANSI*), el va fi înlocuit la returnarea numelui găsit cu caracterul "_". Acest caracter este același în toate paginile de cod *OEM* și *ANSI*.

Convențiile de nume și numele lungi

O aplicație permite utilizatorului să specifice numele lung care să fie atribuit unui fișier sau unui director, dar nu-i permite să-și aleagă și numele scurt al acestuia. Aceasta se datorează faptului că spațiul numelor scurte și spațiul numelor lungi sunt considerate ca fiind un singur spațiu unificat. După cum se poate observa, nu este posibil să avem două fișiere cu același nume lung și cu nume scurte diferite. Această restricție a fost impusă pentru a preveni confuzia din rândul utilizatorilor, și aplicațiilor, în ceea ce privește numele propriu al fișierului sau directorului. Pentru transparența aceastei restricții, atunci când avem un nume lung de fișier sau director, și dorim să-l creăm, aplicația verifică dacă există numele lung și dacă nu există, generează automat numele scurt atașat numelui lung, pe baza numelui lung, astfel încât să nu existe alt fișier/director care să aibă același nume scurt cu cel care va fi creat.

Tehnica aleasă pentru generarea automată a numelor scurte din nume lungi este preluată și adaptată din *Windows NT*. Numele scurt generat automat este compus din două părți: ***numele de bază*** și ***coada numerică***.

Efectul intrărilor cu nume lung în directoare asupra versiunilor precedente de FAT

Suportarea numelor lungi de fișiere este foarte importantă pentru discul hard. Numele lungi sunt suportate și de mediile de date portabile (*removable media*). Implementarea permite suportul pentru nume lungi de fișiere, fără să încalce compatibilitatea cu formatul existent de *FAT*. Un disc poate fi citit de un sistem mai vechi fără probleme de compatibilitate. Un disc deja creat, nu trebuie să fie transformat pentru a suporta numele lungi. Toate fișierele existente rămân neschimbate. Intrările cu nume lung din directoare sunt adăugate pe măsură ce se creează fișiere/directoare cu nume lung.

Intrările cu nume lung sunt ca fișierele ascunse sau ca și cele ascunse sau sistem, pentru sistem *down-level*. Acestea sunt suficiente pentru a nu cauza probleme utilizatorului obișnuit. Utilizatorul poate muta fișierele folosind formatul 8.3 pe alt disc și invers, fără efecte secundare.

Este interesant ce se întâmplă pe disc cu un sistem de fișiere *FAT down-level* în momentul în care este schimbat directorul curent. Acest lucru poate afecta intrările cu nume lung, deoarece sistemele *down-level* ignoră aceste intrări și nu se asigură dacă sunt asociate corect numelor scurte.

Un sistem *down-level* vede aceste intrări numai atunci când caută eticheta de volum a discului. Pe un sistem *down-level*, eticheta de volum poate fi returnată incorect în cazul în care aceasta nu apare înaintea tuturor intrărilor cu nume lung în directorul rădăcină. Acest lucru se datorează faptului că aceste intrări au atributul de volum setat.

Dacă se încearcă ștergerea etichetei de volum, s-ar putea ca o intrare cu nume lung să fie ștearsă. Numele lung nu va mai fi valid deoarece unele dintre intrările sale sunt marcate ca fiind șterse.

Dacă un fișier este redenumit pe un sistem *down-level*, atunci numai numele scurt va fi redenumit. În acest caz numele lung nu va fi afectat, dar datorită faptului că prin redenumire se schimbă suma de control, intrările cu nume lung vor deveni invalide. Această redenumire nu trebuie să intre în conflict cu nici un nume lung. Altfel, un sistem *down-level* poate crea un nume scurt într-un fișier care este identic cu un nume al altui fișier, atunci când nu se ține seama de tipul literelor. Dacă fișierul este șters, atunci numele lung al acestuia, este un simplu orfan. Dacă sunt create câteva fișiere noi, numele lung poate fi incorect asociat cu noul fișier. Ca și în cazul redenumirii, suma de control ajută la prevenirea acestor asocieri.

Alte informații cu privire la directoarele FAT

Intrările în director cu nume lung, sunt identice pe toate tipurile de *FAT*. Pentru volumele *FAT32*, *driver*-ul sistemului de fișiere nu trebuie să permită crearea unui lanț de *cluster*-e mai mare de `0x100000000` bytes, și ultimul byte al lanțului care are această mărime nu poate fi alocat fișierului. Din acest motiv, nici un fișier nu poate avea mărimea mai mare decât `0xFFFFFFFF` bytes. Aceasta este limita fundamentală pe toate tipurile de *FAT*.

De asemenea, *driver*-ul unui sistem de fișiere *FAT* nu trebuie să permită unui director (un fișier care este de fapt un *container* pentru alte fișiere) să fie mai mare de `65536 * 32 = 2.097.152` bytes.

Această limită nu se aplică numărului de fișiere din director, ci reprezintă mărimea directorului și nu are nimic în comun cu conținutul directorului. Există două motive întemeiate pentru această limită:

1. Deoarece directoarele nu sunt sortate sau indexate, nu este recomandabilă crearea de directoare uriașe (care conțin foarte multe fișiere). În cazul directoarelor mari, operațiile cum ar fi cea de adăugare (care verifică toate intrările din director, pentru a vedea dacă intrarea ce urmează a fi creată mai există sau nu) sunt foarte ineficiente.
2. Sunt multe *driver*-e pentru sistemele de fișiere *FAT* și utilitare de disc, inclusiv cele de la *Microsoft*, care nu sunt capabile să citească atât de multe intrări în director, deoarece numărul de intrări din director este stocat folosind o variabilă reprezentată pe 16 biți.

Bibliografie

- **www.microsoft.com**

Ciprian Ileană și Claudiu Soroiu sunt studenți în anul I la Universitatea Babeș-Bolyai din Cluj. Pot fi contactați la adresele **ciprian_il@yahoo.com**, *respectiv* **csoroiu@yahoo.com**.

COMPUTER PRESS AGORA

OFERTĂ SPECIALĂ!

Pentru abonamentele încheiate
la revista NetREPORT
în parioada 1 mai–30 iunie
beneficiați de o

REDUCERE DE 65% !

Preț vânzare 12 numere ~~294.000 LEI~~

Abonament 12 numere* **100.000 lei**

*** Cei care sunt abonați deja, pot folosi oferta pentru a-și prelungi abonamentul**

CONECTAT LA VIITORUL DIGITAL

COMPUTER PRESS AGORA str. Tudor Vladimirescu nr. 63/1, Tg. Mureș (CP 230/1), 4300
Tel. 065-166 516, Fax 065-166 290, e-mail: office@agora.ro

Datorită noului Armada M700 cu procesor Intel® Pentium® III 750 MHz poți acum să faci față oricăror situații neprevăzute. Poți fi în mai multe locuri în același timp și poți face mai multe lucruri deodată. Pentru că Armada M700 are o autonomie de funcționare de peste 5 ore prin folosirea simultană a două baterii iar tastatura Easy Access Internet îți dă posibilitatea conectării imediate la Internet. În plus, cântărește mai puțin de 2,3 kg și are o grosime de numai 3 cm. Astfel că treaba pe care ai început-o la birou o poți termina după program, la cinema. Tehnologia care te inspiră. De la Compaq. Bun venit în noua eră IT.

COMPAQ
Inspiration Technology

EȘTI LA BIROU.
ȘI TOTUȘI NU EȘTI...

COMPAQ ARMADA M700

Procesor Intel® Pentium® III 750MHz • 128MB RAM
• hard disk 12GB • DVD • modem 56k • display TFT 14"
• Ethernet 10/100 • Microsoft Windows 2000/NT4

Calculatoarele Compaq au preinstalat Microsoft® Windows® original.
http://www.microsoft.com/piracy/howtotell

intel inside
pentium® III

scop®
COMPUTERS

Barbu Văcărescu 162
Tel: 231 65 78
Fax: 231 65 96
www.scop.ro

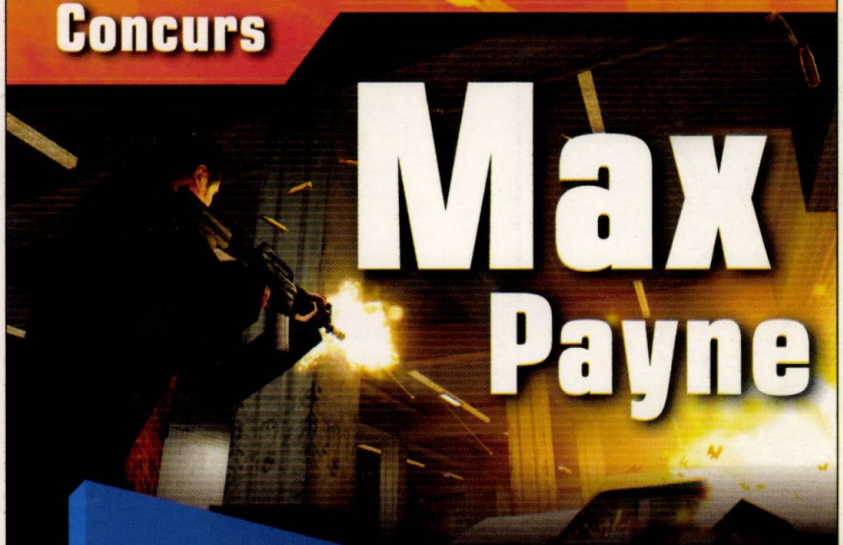
© 2001 Compaq Computer Corporation. COMPAQ and the Compaq logo Registered in U.S. Patent and Trademark Office. Armada and Inspiration Technology are trademarks of Compaq Information Technologies Group, LP. in the U.S. and other countries. Intel, the Intel Inside Logo and Pentium are registered trademarks of Intel Corporation.

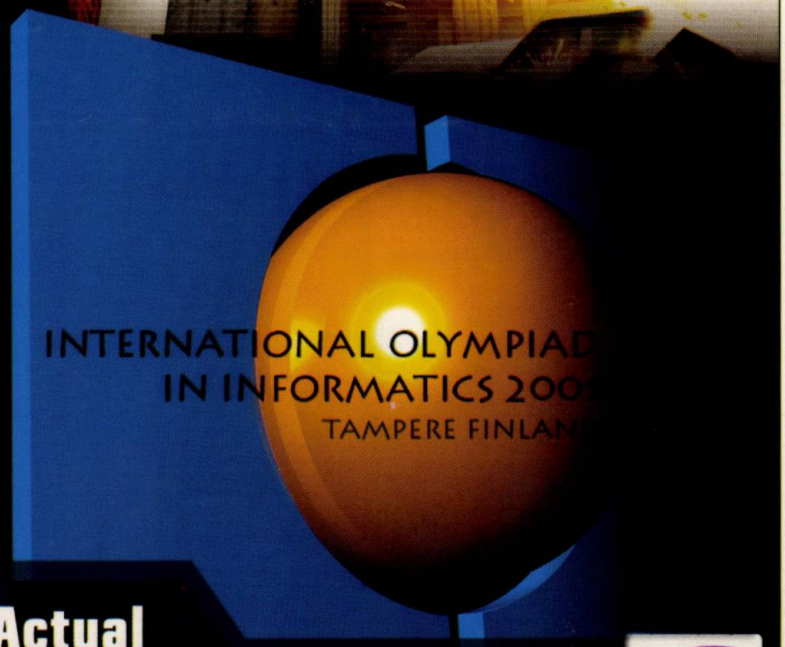
Gazeta de informatica
Revista viitorilor specialiști în tehnologia informației
Octombrie 2001
Vol. 11/6
Lei 20.000

din sumarul numărului:

BURSELE AGORA

Concurs Max Payne





Actual Internet explorer 6

intel pentium 4

intel inside pentium 4

AGORA

pe web

www.agora.ro

www.pcmagazine.ro

www.netreport.ro

www.eweek.ro

www.pcconcrete.ro

news.agora.ro.

www.ginfo.ro

www.aol.ro

conectează-te
ACUM!

Vol. 11, Nr. 6 /2001

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Redactor:
Carmen Șușman (Alba Iulia)
Au mai colaborat:
Oana Georgescu (Brașov)
Sorin Groza (Aiud)
Tiberiu Socaciu (Cluj)
Ionuț Stan (Aiud)
Mihai Stroe (București)
Silviu-Gabriel Udrea (Târgu-Jiu)
Anca Vasilescu (Brașov)
Radu Vișinescu (Ploiești)

Tehnoredactare:
Claudiu Soroiu
Coperta și grafica:
Andrei Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 064/192.422
065/166.516
Fax: 065/166.290
http://www.ginfo.ro

ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359
Editor: Agora Media SRL
Editor al publicațiilor: „eWeek România", „PC Magazine România", „NetReport" și „Gazeta de informatică".
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihai Bucuroiu
Bogdan Manu
Denis Marioara
Florin Tiță
Adriana Pop

Agora MEDIA

Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

Sumar

actual.. PAG. 5

Produsele lunii
Pentium 4 la 2 GHZ
Internet Explorer 6.0
MAX PAYNE
Olimpiada ONLINE

probleme.. PAG. 9

propuse pentru REZOLVARE

Olimpiada Internațională de Informatică

~Tampere, Finlanda~

babel.. PAG. 17

Backtracking în HEAP

Silviu-Gabriel UDREA

concurs.. PAG. 18

Etapa finală a ediției anterioare

Bursele Agora

Talon de participare

Problemele de la prima rundă a noii ediții

focus.. PAG. 34

Un MODEL genetic pentru problema ORARULUI

Radu VIȘINESCU

~algoritmi genetici~

serial.. PAG. 38

Introducere în problematica sistemelor NESECVENȚIALE

Anca VASILESCU
Oana GEORGESCU

~calcul paralel~

mate.. PAG. 43

O posibilă definire ALGEBRICĂ a tipurilor ORDINALE

Tiberiu SOCACIU

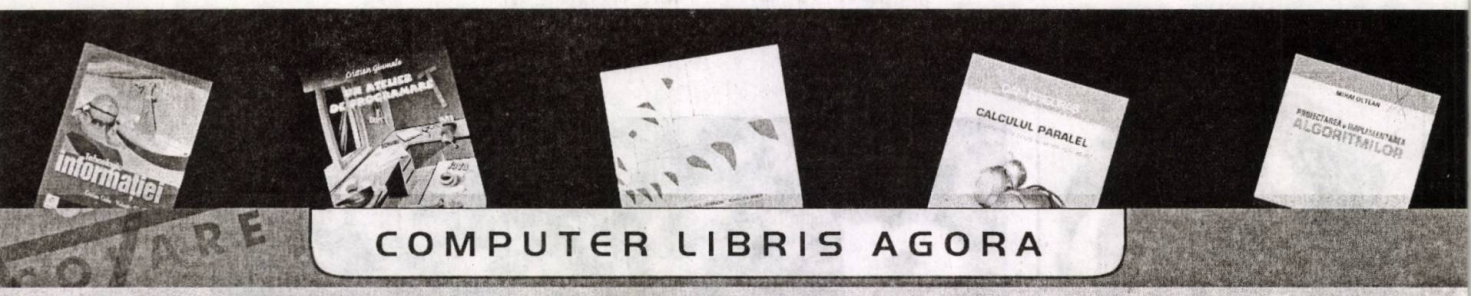
begin.. PAG. 46

La cules de... ALGORITMI

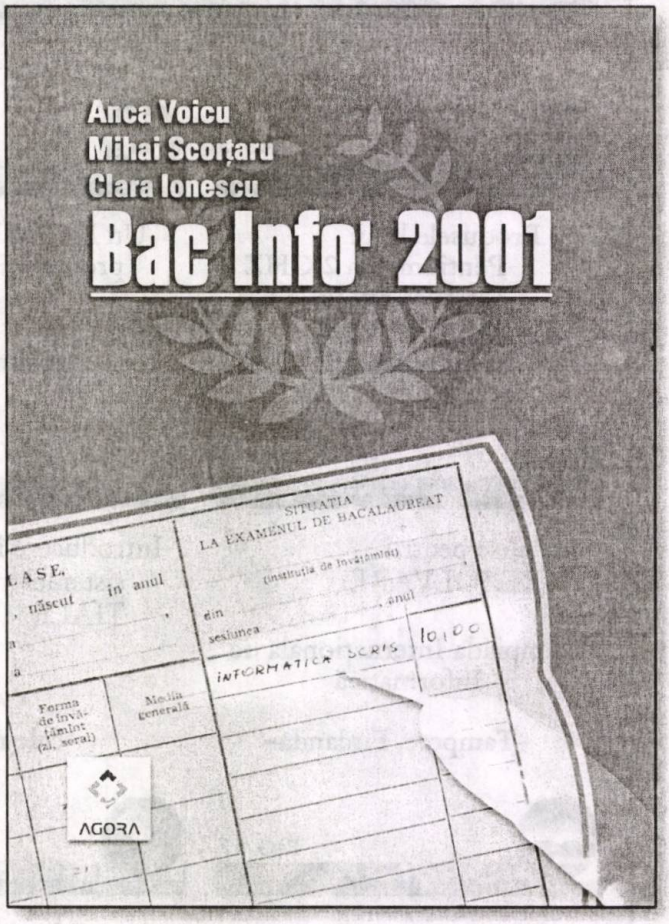
Sorin GROZA
Ionuț STAN

dialog.. PAG. 48

În numărul URMĂTOR



COMPUTER LIBRIS AGORA



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

AGORA

IOI 2001

Olimpiada Internațională de Informatică (I.O.I.), ediția 2001, s-a desfășurat în perioada 14 - 21 iulie la *Tampere*, *Finlanda*. Au participat concurenți din 75 de țări. Delegația *României* a fost condusă de ***prof. univ. dr. Horia Georgescu*** (*A.S.E. București*), ***prof. Emanuela Cerchez*** (*Liceul "Grigore Moisil"*, *Iași*) și ***lect. univ. Clara Ionescu*** (*Universitatea "Babeș-Bolyai"*, *Cluj*).

Gazdele au pus la dispoziție concurenților calculatoare *Osborne Pro*, cu procesor *Pentium III* la 933 MHz și 128 MB RAM. Organizarea a fost excelentă, deși, poate rigidă, după unele gusturi. Evaluarea a fost efectuată foarte rapid, folosindu-se rețeaua existentă.

Problemele au fost, în final, foarte clare și variate. Enunțurile acestora le puteți găsi la pagina 9, urmând ca soluțiile să fie publicate în numărul următor. Comitetul științific local și-a dovedit competența (să nu uităm că renumita firmă *Nokia* își are sediul chiar la *Tampere*).

Concurenții români și-au dovedit din nou valoarea, contribuind la clasarea *României* pe locurile 3-5 în clasamentul pe medalii, după cum urmează:

- ***Mihai Pătrașcu*** *(Craiova)*: medalie de aur (cu al doilea punctaj general);
- ***Vlad Dascălu*** *(Bacău)*: medalie de aur;
- ***Mugurel-Ionuț Andreica*** *(București)*: medalie de argint
- ***András Csaba*** *(Oradea)*: medalie de bronz.

Mihai Pătrașcu, liderul echipei, a avut o comportare excelentă, fiind la un pas de a ocupa locul I absolut. A avut însă "ghinionul" de a concura alături de *Reid Barton* (*Statele Unite ale Americii*), care pe lângă succesele de la IOI., este și figura centrală din istoria *Olimpiadelor Internaționale de Matematică* (patru medalii de aur în patru ani consecutivi, anul acesta cu punctaj maxim, record absolut). De aceea, cred că nu greșim alăturându-l pe *Mihai Pătrașcu* concurenților români care de-a lungul anilor s-au situat pe primul loc în clasamentul general: *Radu Lupșa* și *Mihai Stroe*.

Trebuie remarcat faptul că și în acest an, la fel ca în ultimii nouă ani, echipa de informatică a *României* a obținut rezultate superioare celor de la alte discipline, fapt de multe ori ignorat (cu sau fără intenție).

Pe locurile 1-2 în clasamentul pe medalii s-au situat *Slovacia* și *Singapore*, cu câte două medalii de aur și două de argint. Alături de noi s-au clasat *Bulgaria* și *Coreea de Sud*, tot cu două medalii de aur, una de argint și una de bronz.

Au avut loc schimbări spectaculoase în ierarhia considerată tradițională. *Rusia*, clasată pe primul loc în anul 2000, a coborât spectaculos pe un loc nedorit. A crescut la fel de spectaculos valoarea concurenților din *Singapore*, *Sri Lanka* (!), *Israel* etc.

În acest moment, cu mici excepții (*Albania*, *Mozambic* etc., care n-au obținut nici un punct), nu mai putem să vorbim de favoriți certi și de reprezentative condamnate definitiv la ultimele locuri. Să se datoreze rezultatele deosebite ale unor țări "mici" investițiilor masive pe care le fac în informatică?

În contextul acestor schimbări radicale în clasament, trebuie remarcată evoluția deosebită a echipei României, situată în ultimii ani, în mod constant, pe una dintre primele cinci poziții.

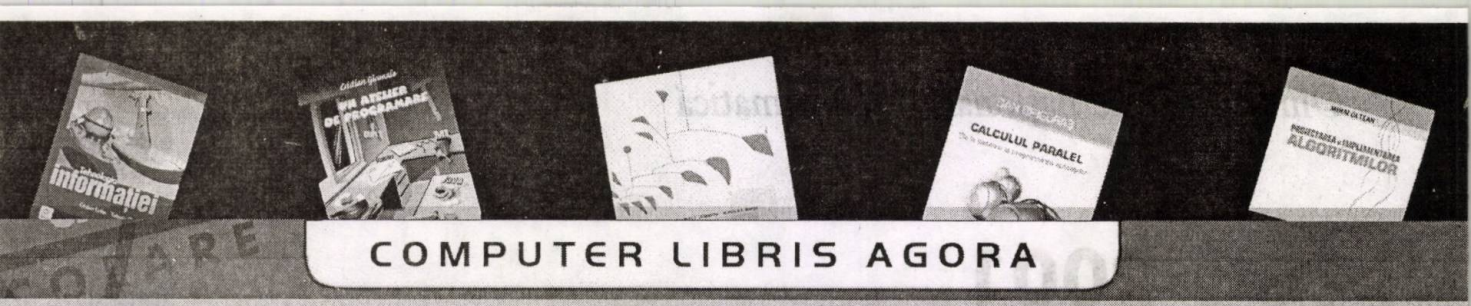
Considerăm că această evoluție se datorează atracției spre informatică a celor mai strălucițti elevi ai noștri, dar și prezenței unui corp profesoral de o valoare deosebită, dublată de mult entuziasm!

Participarea la o olimpiadă internațională trebuie să fie și un prilej de a cunoaște, măcar în fugă, și aspecte legate de țara gazdă. Iată câteva informații:

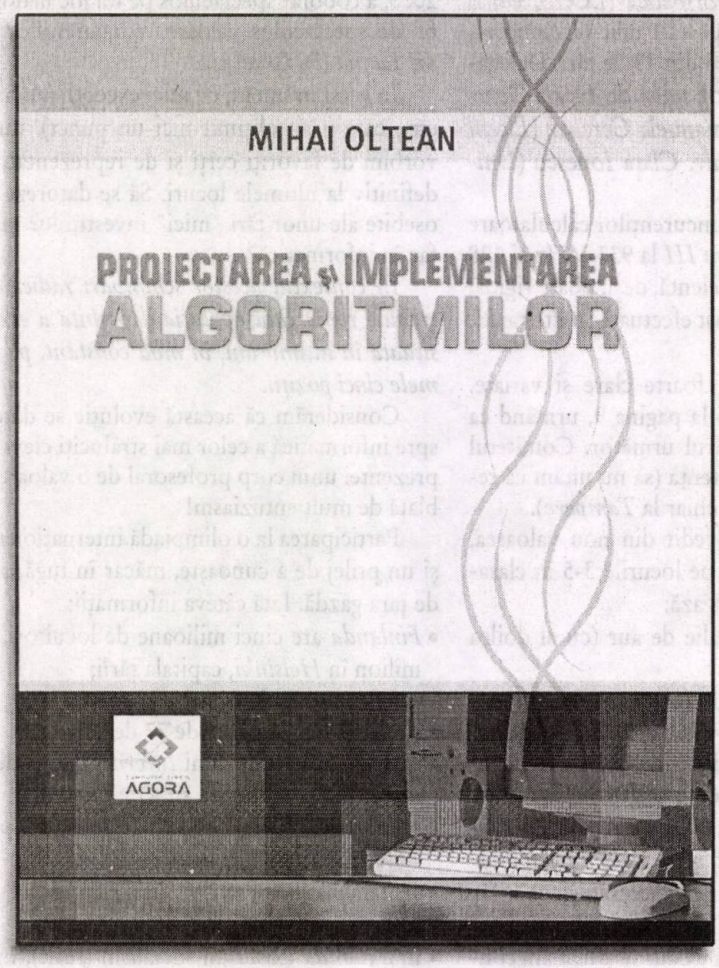
- *Finlanda* are cinci milioane de locuitori, dintre care un milion în *Helsinki*, capitala țării;
- există 188.000 de lacuri;
- speranța de viață este de 77 de ani;
- există 148 de conexiuni *Internet* la mia de locuitori și 75 de telefoane mobile la suta de locuitori;
- salariul mediu lunar este de 2300 EURO pentru bărbați și 1900 EURO pentru femei;
- săptămâna de lucru este, în medie, de 38,6 ore;
- rata șomajului este de 9,5%;
- în *Finlanda* există un socialism (real!), în care nu există discrepanțe mari între cei "săraci" și cei "bogați"; impozitarea crește proporțional cu venitul global, dar se referă și la diferite taxe; de exemplu, amenzile de circulație sunt proporționale cu venitul global (în acest an directorul firmei *Nokia* a fost amendat cu 15.000$ pentru că a trecut pe culoarea roșie a semaforului!);
- susținerea învățământului este o politică de stat; de exemplu, studenții primesc automat aproximativ 500 EURO în fiecare lună, iar pentru doctoranzi (mulți dintre ei români) suma se ridică la 1100 EURO.

IOI 2001 s-a încheiat cu succes pentru *România*, dar nu a fost singurul concurs de la care echipa țării noastre s-a întors cu multe medalii. Detalii despre celelalte olimpiade internaționale veți putea citi în numerele următoare.

Horia Georgescu



COMPUTER LIBRIS AGORA



"Cum aș putea să justific existența acestei cărți? Cum aș putea spre exemplu să justfic cele 100 de pagini despre programarea dinamică, acum când această metodă nu mai figurează în programele școlare? Nu știu. Sau, cum aș putea să justific capitolul despre algoritmi genetici, când pe plan mondial abia de câțiva ani a fost dovedită utilitatea și eficiența lor? Din nou nu știu! Dar rămâne bănuiala: a scrie algoritmi înseamnă a crea. Și nu poți crea fără să gândești."

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Intel Pentium 4 la 2 GHz

Pe data de 27 august 2001 compania Intel a lansat primul procesor pentru calculatoarele destinate utilizatorilor obișnuiți, care are o frecvență de 2 GHz. Chiar dacă procesorul nu aduce nici o altă noutate importantă, atingerea acestei bariere psihologice îi dă dreptul de a deveni produsul lunii.

Istoria Pentium 4

Procesorul *Intel Pentium 4* a apărut pe piață la sfârșitul lunii noiembrie a anului 2000. Atunci existau variante cu frecvențe de 1,4 și 1,5 GHz.

Pentium 4 a fost conceput în așa fel încât să permită obținerea unor frecvențe mari, chiar dacă valoarea frecvenței nu reflectă, întru totul, performanța unui procesor. Totuși, un mare număr dintre cei care cumpără procesoare sunt interesați, în primul rând, de această valoare.

Astfel, *Pentium 4* avea frecvențe mai mari decât procesoarele *Athlon* de la firma rivală *AMD*, dar performanțele erau mai mici.

În luna ianuarie a acestui an a apărut și o variantă cu frecvența de 1,3 GHz, deoarece cel mai rapid procesor *Pentium III* avea frecvența de doar 1 GHz și era nevoie de procesoare cu frecvențe intermediare (între 1 GHz și 1,4 GHz).

A urmat apoi frecvența de 1,7 GHz, chiar dacă următorul salt logic ar fi fost la 1,6 GHz. După câteva luni a apărut și procesorul *Pentium 4* cu frecvența de 1,6 GHz, împreună cu cel care avea frecvența de 1,8 GHz.

În sfârșit, pe data de 27 august, au apărut procesoare cu frecvențe de 1,9 și 2 GHz.

Se pare că versiunea la 2 GHz va fi ultima din această familie care va folosi tehnologia de 0,18 microni. Se preconizează trecerea la 0,13 microni; viitorul *Pentium 4* care va folosi această tehnologie este cunoscut acum sub numele de cod *Northwood*. Acesta va fi lansat în curând și va avea la început frecvențe de până la 2,2 GHz.

Lupta cu gigahertzii

Intel și *AMD* au început în 1999 o cursă pentru atingerea barierei de 1 GHz. *Intel* a pierdut atunci bătălia, deoarece pe data de 6 martie 2000 a fost lansat pe piață un procesor *AMD Athlon* care avea o frecvență de 1 GHz. Chiar dacă procesorul *Pentium III* care funcționa la aceeași frecvență a apărut doar cu două zile mai târziu (8 martie), *Intel* a trebuit să se recunoască învinsă în această luptă.

Următoarea barieră psihologică era cea de 2 GHz. De această dată *AMD* nu a putut lupta cu *Intel* datorită faptului că procesorul *Athlon* nu era proiectat pentru obținerea de frecvențe mari în detrimentul performanțelor. Astfel, în momentul în care *Intel* reușea să atingă această frecvență, cel mai rapid procesor *Athlon* nu funcționa decât la 1,4 GHz.

Perfomanțe

În prima jumătate a anului, cel mai rapid procesor *Athlon* de la *AMD* depășea în majoritatea testelor cel mai rapid procesor de la *Intel*, chiar dacă funcționa la o frecvență mai mică.

Se spunea că e doar o problemă de timp până în momentul în care frecvența mai mare a procesoarelor *Intel* își va spune cuvântul.

Unii susțin că la 2 GHz, *Pentium 4* a reușit, în sfârșit, să depășească procesorul *Athlon*, dar părerile sunt împărțite. Totuși, e clar că un *Pentium 4* la 2 GHz depășește un *Athlon* la 1,4 GHz în majoritatea testelor, dar diferențele nu sunt semnificative.

Se spune că *Intel* are din nou cel mai rapid procesor de pe piață și din punct de vedere al performanțelor reale, dar se pare că situația se va schimba în scurt timp. *AMD* plănuiește lansarea unui nou procesor *Athlon*, probabil sub denumirea *Athlon XP*, care va depăși din nou procesorul *Pentium 4*, chiar dacă frecvența va fi mai mică.

Concluzii

Atingerea frecvenței de 2 GHz este, fără îndoială, un eveniment demn de luat în seamă. Totuși, această lansare nu are decât un efect psihologic, deoarece nu aduce nici o noutate tehnologică.

Intel a reușit să ajungă prima la această frecvență, dar se pune problema oportunității acestei lansări. La ora actuală nu există nici o aplicație care să necesite folosirea unui astfel de procesor. Pentru majoritatea utilizatorilor este suficient un procesor de 1 GHz, sau chiar mai puțin. Cei care pot beneficia cu adevărat de performanțele acestui procesor sunt puțini. Totuși, datorită lansării acestui procesor, *Intel* a recuperat o parte din prestigiul pierdut în fața celor de la *AMD*.

Internet Explorer 6.0

Compania Microsoft a lansat la sfârșitul lunii august o nouă versiune a popularului său program de navigare Internet Explorer. Chiar dacă nu aduce noutăți semnificative, acest nou browser are o importanță deosebită, în primul rând datorită integrării sale în viitorul sistem de operare Windows XP.

Războiul browser-elor

Până acum câțiva ani, piața programelor de navigare era dominată de *browser*-ul *Navigator* produs de *Netscape*. În momentul lansării sistemului de operare *Windows '95 OSR 2*, firma concurentă *Microsoft* a oferit gratuit propriul program de navigare: *Internet Explorer*.

Din acel moment produsul oferit de *Microsoft* a început să acapareze piața programelor de navigare, ajungându-se acum în situația în care mai mult de 75% dintre cei care navighează pe *Web* folosesc *Internet Explorer*.

Netscape a încercat să revină, dar ultima variantă a produsul său, *Navigator 6.1* nu s-a ridicat la nivelul așteptărilor.

O altă companie care încearcă să pătrundă pe această piață este *Opera*. Această firmă oferă cel mai rapid *browser*, dar facilitățile sale sunt mult mai puține decât cele oferite de rivali. Ultima versiune, *Opera 5.12*, este și ea gratuită, dar în fereastră apare permanent și o reclamă.

Compania *Microsoft* era lideră în războiul *browser*-elor încă de la lansarea versiunii *Internet Explorer 5.0*. Noul program de navigare aduce o serie de îmbunătățiri care consolidează această poziție.

Noutăți

Internet Explorer 6.0 este inclus în sistemele de operare *Windows XP Home Edition* și *Windows XP Professional*,

care vor fi lansate oficial pe data de 25 octombrie. Până atunci, utilizatorii sistemelor de operare *Windows '98*, *Windows '98 Second Edition*, *Windows Me*, *Windows NT 4.0* și *Windows 2000* se pot bucura de noile facilități oferite de programul de navigare de la *Microsoft*.

Securitate

Internet Explorer 6.0 vă oferă unelte necesare pentru a vă întări securitatea în momentul navigării și de a controla informațiile personale pe care le preiau diferitele site-uri *Web*.

Sunt suportate un număr mare de standarde de securitate care permit transferuri securizate de informații și tranzacții financiare folosind rețeaua *Internet*.

De asemenea, sunt oferite facilități de criptare și identificare care asigură securitatea informațiilor personale ale utilizatorilor.

Flexibilitate

Internet Explorer 6.0 vă oferă posibilitatea de a vă personaliza modul de navigare. Au fost adăugate trei noi facilități: *Image Toolbar*, *Media Bar* și *Auto Image Resize*.

Un nou design

Dacă veți folosi sistemul de operare *Windows XP*, veți beneficia de o nouă interfață grafică. Bara de unelte va conține butoane stilizate, fundalul meniurilor va avea culori mai atrăgătoare, vor fi noi căsuțe de dialog, meniuri, bare de rulare etc.

Dezvoltarea de aplicații

Internet Explorer 6.0 ușurează munca celor care dezvoltă aplicații pentru *Web*.

Internet Explorer DHTML Platform oferă facilități *DHTML* pentru realizarea interfețelor aplicațiilor *Web*.

Suportul complet pentru specificațiile *CSS 1* oferă noi posibilități în proiectarea paginilor *Web*. Una dintre cele mai interesante facilități este posibilitatea de a afișa texte pe verticală.

Internet Explorer 6.0 oferă suport complet și pentru specificațiile *DOM 1*, suport care permite aplicațiilor accesarea și actualizarea dinamică a conținutului, stilului și structurii unui document.

Suportul îmbunătățit pentru standardul *SMIL 2.0* oferă posibilitatea creării de documente care au conținut multimedia.

Sunt oferite numeroase alte facilități cum ar fi *mouse wheel events*, suport *XML*, integrare *.NET*, *Java VM* etc.

Chiar dacă noutățile nu sunt impresionante, noul *browser* este mai bun decât predecesorii săi.

MAX PAYNE

Screenshot-urile publicate în ultimii ani promiteau că MAX PAYNE va fi un joc care să merite așteptarea. În sfârșit, jocul a fost lansat și s-a ridicat la nivelul speranțelor gamerilor. Imediat după lansare jocul a ajuns pe primele locuri în toate topurile, motiv pentru care a primit titlul de jocul lunii.

Informații generale

Max Payne a fost produs de compania finlandeză *Remedy Entertainment* și este distribuit de *GodGames*.

Jocul este un *third person shooter* inspirat din filmele de acțiune ale regizorului *John Woo*. Influența filmelor este evidentă, dar aceasta nu diminuează cu nimic plăcerea jucătorului. Singurele dezavantaje ale lui *Max Payne* constau în faptul că jocul este foarte scurt (mai puțin de 10 ore de joc) și nu există mod *multiplayer*.

Totuși, *Max Payne* este un joc de acțiune original, diferit de celelalte *shooter*-e de pe piață și, în majoritatea privințelor, superior acestora.

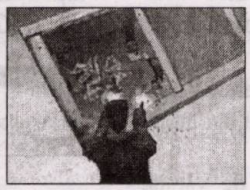
Povestea

Jucătorul este personificat de *Max Payne*, un polițist care lucrează sub acoperire și căruia i-au fost ucise fiica și soția. Personajul va încerca să se răzbune și va ajunge să fie urmărit atât de către polițiști, cât și de mafioți.

Modul de joc

Max Payne poate fi controlat cu ajutorul tastaturii și al *mouse*-ului. Săgețile vor fi folosite pentru deplasare, iar *mouse*-ul pentru a folosi armele. Butonul drept al *mouse*-ului este folosit pentru activarea abilității speciale a lui *Max*, și anume *bullet time*. Folosirea acestei abilități va încetini scurgerea timpului pentru erou, el putându-și anihila mai ușor adversarii care vor fi, practic, neajutorați.

Bullet time-ul este foarte spectaculos. Sunetul armelor de foc pare să vină de departe, se aud bătăile inimii lui *Max* și zgomotul aerului aflat în mișcare. Eroul primește reflexe supraomenești deoarece, chiar dacă scurgerea timpului este încetinită, puteți trage cu armele în același mod în care o faceți fără să folosiți abilitatea specială. De asemenea, pot fi observate gloanțele trase de adversari care se deplasează cu viteză redusă. Abilitatea *bullet time* oferă un avantaj important, dar nu poate fi folosită foarte des, motiv pentru care se recomandă utilizarea ei doar atunci când este absolut necesar.



Max folosește o gamă foarte largă de arme care există și în lumea reală: pistoale, mitraliere, grenade, *cocktail*-uri *Molotov* etc. El poate purta toate armele pe care le găsește, dar muniția este limitată.

Eroul este destul de vulnerabil; un glonț în cap îl poate ucide, iar explozia unei grenade în apropierea sa îi dă puține șanse de supraviețuire. Totuși, el poate ignora aproape complet rănile ușoare prin folosirea așa numitelor *painkillers* pe care le găsește în cele mai diverse locuri.

Jocul folosește un *engine* grafic de ultimă generație care permite obținerea de efecte spectaculoase. Aproape orice poate fi distrus prin focuri de armă, cu excepția unor uși pentru care trebuie găsite chei de acces.

Sistemul necesar

Max Payne necesită un calculator destul de performant. Pentru ca jocul să ruleze impecabil este necesară o frecvență a procesorului de peste 1 GHz, deci se recomandă un *Pentium 4* sau un *Athlon*.

Configurația minimă indicată de producător este un procesor *Pentium II* cu frecvența de 450 MHz sau unul echivalent, 96 MB de memorie RAM, placă video cu accelerator 3D și 16 MB de memorie video, CD-ROM cu viteza 8x și 600 MB de spațiu liber pe disc. Totuși, configurația recomandată necesită un procesor *Athlon* cu frecvența de 1 GHz, 256 MB de memorie RAM și o placă video *GeForce* 2 cu 64 MB.

Recomandări

Jocul *Max Payne* conține numeroase scene violente, motiv pentru care nu este recomandat decât celor care au împlinit 18 ani. În anumite țări a fost interzisă vânzarea sa către minori.

De asemenea, grafica bogată poate avea efecte dăunătoare asupra vederii, motiv pentru care se recomandă o pauză de 15 minute după fiecare oră de joc.

Olimpiada ONLINE

Concursurile de programare online reprezintă un bun prilej de antrenament pentru cei care participă la olimpiadele de informatică. Site-ul acestei luni găzduiește un astfel de concurs. La www.olimpiadaonline.ro veți găsi un consurs online asemănător cu cel organizat de redacția revistei noastre.

Informații generale

Olimpiada online este un concurs care se adresează elevilor de gimnaziu și liceu. Comparabil ca dificultate cu olimpiadele județene și naționale de informatică, concursul permite oricărui elev care are acces la *Internet* să-și evalueze capacitatea de a soluționa diverse probleme de informatică.

Concursul începe pe data de 15 octombrie 2001. În această vară s-a desfășurat o fază experimentală prin care concurenții s-au putut acomoda cu acest concurs.

Există două categorii de vârstă: elevi de gimnaziu și elevi de liceu. Fiecare etapă se desfășoară pe perioada unei luni. În total vor fi cinci etape care se vor desfășura *online*. La început concurenții vor afla enunțurile problemelor care trebuie rezolvate. Rezolvările vor trebui trimise timp de o lună de la data publicării enunțurilor. După cele cinci etape, primii zece clasați la cele două categorii de vârstă se vor califica pentru faza finală care se va desfășura la București. La această etapă toți concurenții vor avea șanse egale, punctajele celor cinci runde anterioare nefiind luate în considerare.

Soluțiile concurenților vor fi corectate în termen de 15 zile de la expirarea termenului de trimitere a soluțiilor.

Evaluarea va fi automată și se va efectua folosind un calculator dotat cu procesor *AMD K6-2* având frecvența de 450 MHz.

Cei mai bine clasați la faza experimentală au fost premiați cu abonamente la revista *PC World* și carduri *PC Net* pentru acces *Internet*.

Premiile oferite la concursul propriu-zis vor consta în calculatoare, imprimante și accesorii. Acestea vor fi oferite de sponsorii principali ai competiției *MDLB Multimedia*, *PC Net* și *IDG*.

Organizatori

Olimpiada online este un concurs organizat de compania *Akela* în colaborare cu *România online*.

Comisia este formată din reprezentanți ai organizatorilor, profesori și studenți care sunt implicați în pregătirea elevilor români care participă la diferite concursuri naționale și internaționale de informatică.

Componența comisiei este următoarea:

- **Stelian Niculescu**, profesor la *Universitatea Politehnica* din București;
- **Rodica Pintea**, profesor la *Liceul Grigore C. Moisil* din București;
- **Lucian Butnaru**, director general la *Akela*;
- **Mihai Bătrâneanu**, director general *PC Net*;
- **Marius Andrei**, student la *Universitatea Politehnica* din București;
- **Mihai Stroe**, student la *Universitatea Politehnica* din București;
- **Cătălin Grigoroșcuță**, student la *Universitatea Politehnica* din București;
- **Eugen Barbu**, student la *Universitatea Politehnica* din București.

Forumul

Organizatorii au creat un forum pentru participanții la concurs.

Aici puteți să adresați întrebări comisiei sau să discutați cu ceilalți participanți. Membrii comisiei vor avea și rol de moderatori ai acestui forum.

Concluzii

Olimpiada online este un concurs de tipul celui organizat de revista noastră. Cei care au participat la concursul organizat de noi nu vor avea dificultăți în a înțelege modul în care se participă la concurs.

Chiar dacă *Olimpiada online* are un regulament propriu, el este foarte asemănător cu cel de la concursul nostru, diferențele fiind neesențiale. Acestea constau în modalitatea de înscriere și modul de trimitere al problemelor. Vă recomandăm să citiți cu atenție regulamentul concursului pentru a evita apariția unor confuzii.

Vă dorim succes!

Informațiile au fost furnizate de organizatorii concursului. Redacția GInfo nu își asumă nici o răspundere pentru corectitudinea acestora.

PROBLEME propuse pentru REZOLVARE

Vă prezentăm enunțurile celor șase probleme propuse spre rezolvare concurenților de la Olimpiada Internațională de Informatică, desfășurată în perioada 14-21 iulie 2001 în orașul finlandez Tampere.

101: Telefoane mobile

În *Tampere* stațiile de telefonie mobilă din generația a patra funcționează după cum urmează.

Teritoriul orașului este împărțit în pătrate. Pătratele formează o matrice de dimensiunea $S \times S$, liniile și coloanele căreia sunt indexate de la 0 la $S - 1$. Fiecare pătrat conține câte o stație de telefonie. Întrucât telefoanele pot fi mutate dintr-o zonă în alta, pot fi pornite sau oprite, numărul de telefoane mobile active din cadrul oricărui pătrat se poate schimba. La cererea stației principale, stațiile din interiorul pătratelor indicate îi transmit, fiecare, numărul curent de telefoane active.

Scrieți un program care recepționează datele transmise de stații și furnizează răspunsuri referitoare la numărul de telefoane mobile active din zonele dreptunghiulare indicate.

Intrări și ieșiri

Datele de intrare sunt citite sub formă de numere întregi de la intrarea standard, iar răspunsurile sunt scrise, de asemenea, ca numere întregi la ieșirea standard.

Datele de intrare sunt codificate după cum urmează. Fiecare linie a fișierului de intrare conține o instrucțiune formată dintr-un cod - un număr întreg (N) - eventual urmat de parametri, care de asemenea sunt numere întregi.

Valorile posibile ale parametrului N sunt 0, 1, 2 și 3. Semnificațiile acestora sunt descrise în continuare.

În cazul în care valoarea lui N este 0, atunci instrucțiunea are un singur parametru (S) și indică inițializarea unei matrice de dimensiunea $S \times S$ ale cărei elemente vor avea toate valoarea 0. Această instrucțiune apare numai o singură dată și întotdeauna este prima în fișierul de in fișierul de intrare.

Dacă valoarea lui N este 1, instrucțiunea are trei parametri (X, Y și A) și indică faptul că numărului de telefoane active din pătratul având coordonatele (X, Y) i se adaugă valoarea A, care poate fi pozitivă sau negativă.

Pentru valoarea 2 a lui N, vom avea patru parametri (L, B, R și T). Instrucțiunea înscrie în fișierul de ieșire suma numerelor de telefoane active din toate pătratele având coordonatele (X, Y), unde $L \le X \le R$, $B \le Y \le T$.

Valoarea 3 a lui N indică terminarea programului. Această instrucțiune apare numai o singură dată și întotdeauna este ultima în fișierul de intrare.

Valorile de la intrare vor fi întotdeauna din intervalul indicat în restricții, deci nu se cere verificarea lor. La fel, orice valoare negativă a numărului A nu va putea conduce la o valoare negativă în matrice. Indexarea pătratelor începe de la 0, prin urmare pentru o matrice de dimensiunea 4×4 avem $0 \le X \le 3$ și $0 \le Y \le 3$.

Programul dumneavoastră nu va înscrie nimic în fișierul de ieșire în cazul instrucțiunilor ce diferă de instrucțiunea având codul 2. Dacă instrucțiunea curentă din fișierul de intrare are codul 2, se cere ca programul să scrie pe câte o linie în fișierul standard de ieșire numărul calculat de telefoane active.

Instrucțiuni de programare

În exemplele ce urmează variabila întreagă `last` conține numărul curent citit din fișierul de intrare, iar variabila întreagă `answer` conține numărul de telefoane calculat de program.

În cazul programelor scrise în C++ și utilizarea *iostreams*, se va folosi următoarea secvență pentru citirea de la intrarea standard și scrierea la ieșirea standard:

```
cin>>last;
cout<<answer<<endl<<flush;
```

În cazul programelor scrise în C sau C++ și utilizarea funcțiilor `scanf()` și `printf()`, se va folosi următoarea secvență pentru citirea de la intrarea standard și scrierea la ieșirea standard:

```
scanf("%d",&last);
printf("%d\n",answer); fflush(stdout);
```

În cazul programelor scrise în *Pascal* se va folosi următoarea secvență pentru citirea de la intrarea standard și scrierea la ieșirea standard:

```
Read(last); ... Readln;
Writeln(answer);
```

Exemplu

Linia	stdin	stdout
1.	0 4	
2.	1 1 2 3	
3.	2 0 0 2 2	
4.		3
5.	1 2	
6.	1 1 2 -1	
7.	2 1 1 2 3	
8.		4
9.	3	

Explicații

- prima linie indică inițializarea unei matrice de dimensiunea 4×4;
- a doua linie adaugă valoarea `+3` la elementul având coordonatele `(1, 2)` din matrice;
- a treia linie calculează suma elementelor din matrice cu indicii (X, Y), unde $0 \le X \le 2$, $0 \le Y \le 2$;
- a patra linie specifică faptul că la ieșirea standard se va afișa valoarea `3`;
- a cincia linie adugă valoarea `+2` la elementul având coordonatele `(1, 1)` din matrice;
- a șasea linie adaugă valoarea `-1` la elementul având coordonatele `(1, 2)` din matrice;
- a șaptea linie calculează suma elementelor din matrice având indicii (X, Y), unde $1 \le X \le 2$, $1 \le Y \le 3$;
- a opta linie specifică faptul că la ieșirea standard se va afișa valoarea `4`;
- a noua linie indică faptul că programul se încheie.

Restricții

- Matricea pătratică va avea cel mult 1024 de linii și cel mult 1024 de coloane: $1 \times 1 \le S \times S \le 1024 \times 1024$.
- În fiecare moment valoarea V corespunzătore unui pătrat va fi un număr întreg pozitiv cu semn care poate fi reprezentat pe 16 biți: $0 \le V \le 2^{15} - 1$ (= 32767).
- Valoarea A va fi un număr întreg cu semn care poate fi reprezentat pe 16 biți; $-2^{15} \le A \le 2^{15} - 1$ (= 32767).
- Numărul U al instrucțiunilor de la intrare va fi cuprins între 3 și 60002: $3 \le U \le 60002$.
- Suma elementelor matricei este cel mult M ($=2^{30}$).

Precizări

- Timpul maxim de execuție admis este de 1 secundă/test.
- Limita maximă a memoriei care poate fi folosită este de 5 MB.
- Au fost 20 de teste, pentru rezolvarea fiecăruia acordându-se cel mult 5 puncte. Pentru 16 dintre ele dimensiunea matricei era de cel mult 512×512.
- Opțiunile de compilare folosite pentru limbajul *Pascal* au fost `-So -O2 -Xs`; opțiunile de compilare folosite pentru limbajele *C* și *C++* au fost `-O2 -static`;
- Facilitățile de testare în rețea au permis ca fișierele de test să fie redirectate la intrarea standard a programului concurentului.

P060102: Jocul Ioiwari

Familia de jocuri *Mancala* cu mărgele și adâncituri într-o tablă de joc este una dintre cele mai vechi forme de distracție. Această problemă prezintă o versiune a jocului, concepută special pentru IOI.

La joc participă doi jucători care folosesc o tablă rotundă, prevăzută cu șapte adâncituri. În plus, fiecare jucător are propria sa bancă de mărgele. Jocul începe prin distribuirea aleatoare a 20 de mărgele în adâncituri; fiecare adâncitură conține cel puțin două și cel mult patru mărgele.

Cei doi jucători mută alternativ. La fiecare mutare, jucătorul alege o adâncitură nevidă, ia toate mărgelele din ea și le păstrează în mână. Apoi, atât timp cât jucătorul mai are mărgele în mână, analizează pe rând următoarele adâncituri, în sensul acelor de ceasornic, începând cu cea care urmează după cea pe care a golit-o. Pentru fiecare dintre aceste adâncituri execută una dintre următoarele operații:

- Dacă are mai mult de o mărgea în mână:
  - dacă în adâncitura curentă există deja cinci mărgele, atunci ia una dintre ele și o pune în banca sa;
  - în caz contrar pune în adâncitură una din mărgelele pe care le ține în mână.

- Dacă are exact o mărgea în mână:
  - dacă adâncitura curentă conține cel puțin una și cel mult patru mărgele, atunci jucătorul mută în bancă toate mărgelele din adâncitură, precum și pe cea din mâna sa;
  - în caz contrar (adâncitura conține 0 sau 5 mărgele), jucătorul plasează mărgeaua din mâna sa în banca jucătorului advers.

Jocul se termină când toate adâncițurile sunt goale. În această situație, câștigătorul este acel jucător care are în bancă cele mai multe mărgele.

Jucătorul care începe jocul are întotdeauna strategie sigură de câștig. Scrieți un program care joacă *Ioiwari* ca prim jucător și câștigă. Adversarul joacă optim, adică o dată ce i se dă ocazia, profită de ea și câștigă jocul.

Intrări și ieșiri

Programul citește datele de intrare de la intrarea standard și scrie la ieșirea standard. Programul este Jucătorul 1, iar ad-

versarul este Jucătorul 2. Programul începe prin citirea de pe prima linie a șapte numere întregi (p_1, ..., p_7) care reprezintă numărul inițial de mărgele din cele șapte adâncituri, în această ordine. Adâncimile sunt etichetate cu numere întregi de la 1 la 7 în sensul acelor de ceasornic. După aceasta, jocul începe cu băncile fără mărgele. Programul trebuie să joace în modul următor:

- dacă este rândul programului să mute, atunci programul va scrie la ieșirea standard eticheta adânciturii pe care o alege ca prim pas al mutării;
- dacă este rândul adversarului să mute, atunci programul vostru va citi de la intrarea standard eticheta adânciturii pe care adversarul o alege ca prim pas al mutării.

Instrumente

Programul jucătorului advers (`ioiwari2` în *Linux*, și `ioiwari2.exe` în *Windows*) joacă, plecând de la o poziție inițială dată, optim ca Jucătorul 2. Acesta va scrie mai întîi la ieșirea standard o primă linie, care va fi citită de programul vostru; această linie descrie numărul inițial de mărgele din adâncituri pentru jocul respectiv. O posibilitate ar fi:

```
4 3 2 4 2 3 2.
```

Apoi, programul va juca încercând să citească mutările Jucătorului 1 de la intrarea standard și scriind propriile mutări la ieșirea standard.

Puteți rula programul vostru și programul `ioiwari2` în ferestre separate și puteți transfera manual conversația între programe. Conversația este memorată în fișierul `ioiwari.out`.

Instrucțiuni de programare

În exemplele următoare, variabila `last` conține cea mai recentă mutare a adversarului, mutarea voastră va apărea în variabila `mymove`.

Dacă programați în *C++* și folosiți *iostreams*, trebuie să implementați citirea de la intrarea standard și scrierea la ieșirea standard exact după cum urmează:

```
cout<<mymove<<endl<<flush;
cin>>last;
```

Dacă programați în *C* sau *C++* și folosiți funcțiile `scanf()` și `printf()`, trebuie să implementați citirea de la intrarea standard și scrierea la ieșirea standard exact după cum urmează:

```
printf("%d\n",mymove); fflush(stdout);
scanf("%d",&last);
```

Dacă programați în *Pascal*, trebuie să implementați citirea de la intrarea standard și scrierea la ieșirea standard exact după cum urmează:

```
Writeln(mymove);
Readln(last);
```

Exemplu

Fie următoarea secvență corectă de șase mutări:

Operație		*Etichetă adâncitură*								
Juc.	*Mutare*	1	2	3	4	5	6	7	*Banca1*	*Banca2*
situația inițială:		4	3	2	4	2	3	2	0	0
1.	2	4	0	3	5	0	3	2	3	0
2.	3	4	0	0	4	1	4	0	3	4
1.	5	4	0	0	4	0	0	0	8	4
2.	4	0	0	0	0	1	1	1	8	9
1.	5	0	0	0	0	0	0	1	10	9
2.	7	0	0	0	0	0	0	0	11	9

Mutările efectuate, conținutul adânciturilor și al băncilor

Modul de punctare

Pentru fiecare test primiți patru puncte dacă programul vostru câștigă jocul, două puncte în caz de remiză, respectiv zero puncte dacă programul vostru pierde jocul.

Precizări

- Timpul maxim de execuție admis este de 1 secundă/test. Programul care reprezintă adversarul este executat ca un proces separat al cărui timp de execuție nu este adunat la timpul de execuție al programului vostru.
- Limita maximă a memoriei care poate fi folosită este de 32 MB.
- Au fost 25 de teste, pentru rezolvarea fiecăruia acordându-se cel mult 4 puncte.
- Opțiunile de compilare folosite pentru limbajul *Pascal* au fost `-So -O2 -Xs`; opțiunile de compilare folosite pentru limbajele *C* și *C++* au fost `-O2 -static`.

P060103: Doi-cinci

Mesajele secrete dintre *Moș Crăciun* și micile sale ajutoare sunt, de obicei, codificate în limbajul 25. Alfabetul limbajului 25 este același cu alfabetul latin, cu o excepție - litera `'Z'` este absentă, adică alfabetul conține cele 25 de litere ale alfabetului latin. Fiecare cuvânt al limbajului 25 este format din exact 25 de litere distincte.

Un cuvânt poate fi scris într-un tablou cu dimensiunea 5 × 5, completând mai întâi liniile; de exemplu, cuvântul `ADJPTBEKQUCGLRVFINSWHMOXY` va fi scris după cum urmează:

```
A  D  J  P  T
B  E  K  Q  U
C  G  L  R  V
F  I  N  S  W
H  M  O  X  Y
```

Un cuvânt corect în limbajul 25 are literele pe fiecare linie și pe fiecare coloană în ordine crescătoare. Astfel, cuvântul `ADJPTBEKQUCGLRVFINSWHMOXY` este un cuvânt corect, spre deosebire de `ADJPTBEGQUCKLRVFINSWHMOXY`, care nu este un cuvânt corect (ordinea crescătoare este încălcată în coloanele a doua și a treia).

Moș Crăciun are un lexicon. Lexiconul conține lista cu toate cuvintele corecte din limbajului 25, în ordine lexicografică crescătoare, însoțite de numerele lor de ordine, începând de la 1.

De exemplu, cuvântul `ABCDEFGHIJKLMNOPQRSTUVWXY` este primul în lexicon, iar `ABCDEFGHIJKLMNOPQRSUTVWXY`, este al doilea cuvânt. Se observă că singura diferență dintre cele două cuvinte este interschimbarea literelor `U` și `T`.

Din nefericire, acest lexicon este uriaș. Scrieți un program care determină numărul de ordine al unui cuvânt dat precum și cuvântul corespunzător unui număr de ordine dat. Există cel mult 2^{31} cuvinte în lexicon.

Intrarea

Numele fișierului de intrare este `TWOFIVE.IN` și conține două linii. Pe prima linie apare una dintre literele `'W'` sau `'N'`.

- Dacă pe prima linie apare `'W'`, atunci a doua linie conține un cuvânt corect din limbajul 25.
- Dacă pe prima linie apare `'N'`, atunci a doua linie conține numărul de ordine al unui cuvânt corect existent în limbajului 25.

Ieșirea

Numele fișierului de ieșire este `twofive.out` și conține o singură linie.

- Dacă pe a doua linie a fișierului de intrare apare un cuvânt, atunci linia fișierului de ieșire trebuie să conțină numărul de ordine al acestui cuvânt.
- Dacă pe a doua linie a fișierului de intrare apare un număr, atunci linia fișierului de ieșire trebuie să conțină cuvântul care are acel număr de ordine.

Exemple

`twofive.in`

```
ABCDEFGHIJKLMNOPQRSUTVWXY
```

`twofive.out`

```
2
```

`twofive.in`

```
2
```

`twofive.out`

```
ABCDEFGHIJKLMNOPQRSUTVWXY
```

Precizări

- Timpul maxim de execuție admis este de 0.02 secunde/test.
- Limita maximă a memoriei care poate fi folosită este de 32 MB.
- Au fost 20 de teste, pentru rezolvarea fiecăruia acordându-se cel mult 5 puncte.
- Opțiunile de compilare folosite pentru limbajul *Pascal* au fost `-So -O2 -XS`; opțiunile de compilare folosite pentru limbajele *C* și *C++* au fost `-O2 -static`.

P060104: Score

Score este un joc pentru doi jucători care mută un jeton dintr-o poziție în alta a unei table. Tabla are N poziții, numerotate de la 1 la N și un număr de săgeți. Fiecare săgeată pleacă dintr-o poziție către o altă poziție. Fiecare poziție este proprietatea unui singur jucător, numit proprietarul poziției respective. În plus, fiecărei poziții i se asociază o valoare întreagă pozitivă. Valorile asociate pozițiilor sunt distincte. Poziția 1 este poziția de start. Inițial, ambii jucători au scorul 0.

Jocul se desfășoară în modul următor. Notăm cu C poziția curentă a jetonului la începutul mutării. La începutul jocului, poziția curentă C este poziția 1. O mutare constă din următoarele operații:

- Dacă valoarea lui C este mai mare decât scorul curent al proprietarului poziției C, atunci valoarea acestei poziții devine noul scor al proprietarului poziției C; în caz contrar, scorul proprietarului poziției C nu se modifică. Scorul celuilalt jucător nu se modifică în nici unul dintre cazurile de mai sus.
- Proprietarul poziției C alege una dintre săgețile care pleacă din poziția curentă a jetonului și mută jetonul în poziția în care se află vârful săgeții.

Se observă faptul că un anumit jucător poate efectua mai multe mutări consecutive.

Jocul se termină atunci când jetonul revine în poziția de start. Câștigă jucătorul care la terminarea jocului are cel mai mare scor.

Săgețile sunt întotdeauna aranjate astfel încât:

- întotdeauna din poziția curentă a jetonului pleacă cel puțin o săgeată;
- fiecare poziție P este accesibilă din poziția de start, adică există o succesiune de săgeți de la poziția de start la poziția P;
- se garantează că jocul se termină după un număr finit de mutări.

Scrieți un program care să joace acest joc și să îl câștige. Toate jocurile propuse în evaluare vă permit să câștigați, indiferent dacă începeți sau nu primul. Jucătorul advers din evaluare joacă optimal, în sensul că dacă i se oferă o șansă de câștig, atunci va profita de ea și va câștiga jocul.

Intrări și ieșiri

Programul vostru citește de la intrarea standard și scrie la ieșirea standard. Programul vostru este Jucătorul 1, iar adversarul este Jucătorul 2. La început, vor trebui citite următoarele date de la intrarea standard:

- prima linie conține un număr întreg N care reprezintă numărul pozițiilor ($1 \le N \le 1000$);
- fiecare dintre următoarele N linii conțin câte N numere întregi care descriu săgețile astfel: dacă există o săgeată de la poziția i la poziția j, atunci cel de-al j-lea număr de pe a i-a linie este 1, iar dacă o astfel de săgeată nu există, atunci numărul respectiv va fi 0;

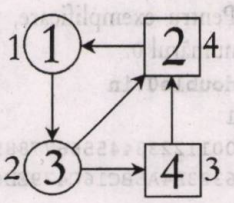
- următoarea linie conține N numere întregi care indică proprietarii pozițiilor; dacă poziția i îl are ca proprietar pe Jucătorul 1 (programul vostru), atunci cel de-al i-lea întreg este 1, iar în caz contrar este 2.
- următoarea linie conține N numere întregi care indică valorile pozițiilor; dacă cel de-al i-lea întreg este j, atunci valoarea asociată poziției i este j; toate valorile j satisfac relația $1 \le j \le N$ și sunt distincte.

După citirea acestor date jocul va începe, iar jetonul se află în poziția 1.

Programul se va încheia în momentul în care jetonul revine în poziția 1 și se desfășoară astfel:

- Dacă este rândul programului vostru să mute, atunci el va scrie la ieșirea standard numărul P corespunzător poziției următoare ($1 \le P \le N$).
- Dacă este rândul adversarului să mute, atunci programul vostru trebuie să citească de la intrarea standard numărul P corespunzător poziției următoare ($1 \le P \le N$).

Să considerăm tabla din figură. Pozițiile încadrate într-un cerc aparțin Jucătorului 1, iar cele încadrate într-un pătrat aparțin Jucătorului 2.



Valoarea fiecărei poziții este scrisă în interiorul cercului sau pătratului corespunzător, iar numărul poziției este precizat lângă pătrat sau cerc. De exemplu, un joc se desfășoară după cum urmează:

Exemplu

Linia	**stdin**	**stdout**
1.	4	
2.	0 1 0 0	
3.	0 0 1 1	
4.	0 0 0 1	
5.	1 0 0 0	
6.	1 1 2 2	
7.	1 3 4 2	
8.		2
9.		4
10.	1	

Explicații

- prima linie a intrării indică faptul că tabla are patru poziții;
- a doua linie indică faptul că din prima poziție pleacă o singură săgeată care are vârful în a doua poziție;
- a treia linie arată că din a doua poziție pleacă două săgeți care au vârfurile în a doua și a treia poziție;
- conform celei de-a patra linii, din a treia poziție pleacă o singură săgeată care are vârful în a patra poziție;
- cea de-a cincea linie arată că din a patra poziție pleacă tot o săgeată care are vârful în prima poziție;
- proprietarii celor patru poziții sunt precizați în cea de-a șasea linie: primul jucător deține prima și a doua poziție, iar al doilea deține a treia și a patra poziție;
- valorile celor patru poziții sunt indicate în cea de-a șaptea linie;
- prima linie a ieșirii (linia a opta) indică faptul că primul jucător mută jetonul în a doua poziție;
- cea de-a doua linie a ieșirii (linia a noua) indică faptul că primul jucător mută jetonul în a patra poziție;
- ultima linie a intrării (linia a zecea) indică faptul că al doilea jucător a mutat jetonul în prima poziție și jocul se încheie.

La sfârșitul jocului, Jucătorul 1 are scorul 3, iar Jucătorul 2 are scorul 2, deci Jucătorul 1 câștigă.

Instrucțiuni de programare

În exemplele următoare `target` este o variabilă care memorează poziția curentă.

Dacă programați în C++ și utilizați *iostreams*, trebuie să implementați citirea de la intrarea standard și scrierea la ieșirea standard astfel:

```
cin>>target;
cout<<target<<endl<<flush;
```

Dacă programați în C sau C++ și utilizați funcțiile `scanf()` și `printf()`, trebuie să implementați citirea de la intrarea standard și scrierea la ieșirea standard astfel:

```
scanf("%d",&target);
printf("%d\n",target); fflush(stdout);
```

Dacă programați în *Pascal*, trebuie să implementați citirea de la intrarea standard și scrierea la ieșirea standard astfel:

```
Writeln(target);
Readln(target);
```

Instrumente

Aveți la dispoziție un program (`score2` pentru sistemul de operare *Linux* și `score2.exe` pentru sistemul de operare *Windows*).

Programul citește din fișierul `score.in` descrierea jocului în formatul prezentat anterior și scrie această informație la ieșirea standard folosind același format. Această ieșire poate fi folosită de programul vostru ca intrare pentru testare.

În continuare programul joacă folosind o strategie aleatoare citind mutările programului vostru de la intrarea standard și scriind propriile lui mutări la ieșirea standard.

Punctaj și evaluare

Pentru fiecare test la care programul vostru câștigă jocul, veți primi punctele corespunzătoare. Dacă jocul este pierdut nu veți primi nici un punct. În timpul evaluării progra-

mul vostru va juca împotriva unui alt program având la dispoziție o limită de timp cu o secundă mai mare decât cea de la evaluarea propriu-zisă. Intrările și ieșirile programului sunt păstrate, iar apoi programul este executat pentru a doua oară, dar intrarea va fi redirectată spre un fișier. În acest moment va fi măsurat timpul oficial de execuție. La această a doua execuție programul trebuie să furnizeze aceleași ieșiri pe care le-a furnizat la prima execuție.

Precizări

- Timpul maxim de execuție admis este de 1 secundă/test.
- Limita maximă a memoriei care poate fi folosită este de 32 MB.
- Au fost 20 de teste, pentru rezolvarea fiecăruia acordându-se cel mult 5 puncte.
- Opțiunile de compilare folosite pentru limbajul *Pascal* au fost `-So -O2 -Xs`; opțiunile de compilare folosite pentru limbajele *C* și *C++* au fost `-O2 -static`.

P060105: Criptare dublă

Standardul avansat de criptare (*AES - Advanced Encryption Standard*) utilizează un algoritm nou și puternic. Acest algoritm operează cu trei blocuri de câte 128 de biți. Pentru un bloc necriptat p (textul inițial) și un bloc cheie k, funcția de criptare E returnează un bloc criptat c (textul criptat):

$$c = E(p, k).$$

Funcția de decriptare D este inversa funcției de criptare E:

$$D(E(p, k), k) = p,$$
$$E(D(c, k), k) = c.$$

În standardul *Dublu AES*, două blocuri cheie independente k_1 și k_2 sunt utilizate consecutiv, mai întâi k_1 și apoi k_2:

$$c_2 = E(E(p, k_1), k_2).$$

În această problemă se furnizează și un număr întreg s. Doar cei $4 \cdot s$ biți din stânga ai cheilor sunt relevanți, ceilalți biți (cei $128 - 4 \cdot s$ din dreapta cheii) având toți valoarea 0.

Se cere să reconstituiți perechea de chei de criptare pentru unele blocuri criptate cu *Dublu AES*. Se cunosc blocul inițial p, blocul criptat c_2 și numărul întreg s cu semnificația de mai sus.

Algoritmul *AES* de criptare și algoritmul *AES* de decriptare sunt disponibili într-o bibliotecă.

Nu va trebui să realizați un program care rezolvă problema, ci doar să descoperiți perechile de chei corespunzătoare seturilor de date.

Intrare

Se dau zece instanțe ale problemei și anume fișierele text `double1.in`, `double2.in`, ..., `double10.in`. Fiecare fișier este format din trei linii. Prima linie conține numărul întreg s, linia a doua conține blocul necriptat p, iar linia a treia conține blocul criptat c_2 obținut din p cu ajutorul criptării *Dublu AES*. Ambele blocuri sunt citite ca șiruri de caractere ce conțin 32 de cifre hexazecimale (`'0'..'9'`, `'A'..'F'`). Biblioteca care va fi prezentată în continuare vă pune la dispoziție o rutină pentru transformarea șirurilor de caractere în blocuri. Toate fișierele de intrare admit soluții.

Ieșirea

Trebuie să creați zece fișiere de ieșire, corespunzătoare fișierelor de intrare date. Fiecare fișier de ieșire conține trei linii: pe prima linie se va afla textul `#FILE double I`, unde `I` este numărul fișierului de intrare, pe a doua linie se află cheia k_1, iar linia a treia cheia k_2, astfel încât

$$c_2 = E(E(p, k_1), k_2).$$

Ambele blocuri trebuie scrise ca șiruri de caractere ce conțin 32 de cifre hexazecimale (`'0'..'9'`, `'A'..'F'`). Biblioteca vă pune la dispoziție o rutină pentru transformarea blocurilor în șiruri de caractere. Dacă există mai multe soluții, atunci va fi generată una singură.

Exemplu

Pentru exemplificare, vom folosi fișierul de intrare cu numărul `0`.

double0.in

```
1
00112233445566778899AABBCCDDEEFF
6323B4A5BC16C479ED6D94F5B58FF0C2
```

Un posibil conținut al fișierului de ieșire ar putea fi:

```
#FILE double 0
A0000000000000000000000000000000
70000000000000000000000000000000
```

Biblioteca

Partea declarativă a bibliotecii pentru limbajul *FreePascal* este următoarea:

```
type HexStr=string[32]; { '0'..'9', 'A'..'F' }
     Block=array[0..15] of Byte; { 128 bits }

procedure HexStrToBlock(const hs:HexStr;
                              var b:Block);

procedure BlockToHexStr(const b:Block;
                            var hs:HexStr);

procedure Encrypt(const p,k:Block;
                              var c:Block);
  { c = E(p,k) }

procedure Decrypt(const c,k:Block;
                              var p:Block);
  { p = D(c,k) }
```

Programul `aestoolp.pas` ilustrează modul de utilizare a bibliotecii *FreePascal*.

Partea declarativă a bibliotecii pentru *GNU C/C++* este următoarea:

```
typedef char HexStr[33];
      // '0'..'9', 'A'..'F', '\0'-terminator

typedef unsigned char Block[16];  // 128 bits

void hexstr2block(const HexStr hs, Block b);

void block2hexstr(const Block b, HexStr hs);

void encrypt(const Block p, const Block k,
                                    Block c);
  // c = E(p,k)

void decrypt(const Block c, const Block k,
                                    Block p);
  // p = D(c,k)
```

Programul `aestoolp.c` ilustrează modul de utilizare a bibliotecii *GNU C/C++*.

Restricție

- Numărul s al cifrelor hexazecimale semnificative din orice cheie este cuprins între 1 și 5 ($1 \le s \le 5$).

Precizări

- Au fost 10 de teste; pentru rezolvarea primului test s-a acordat cel mult un punct, iar pentru celelalte nouă s-au acordat cel mult câte 11 puncte.
- Un program eficient poate reconstitui cheile pentru orice fișier corect de intrare în cel mult 10 secunde.

060106: Depozit

O firmă finlandeză de *high technology* are un depozit dreptunghiular uriaș. La depozit lucrează un muncitor și un director. Laturile depozitului sunt denumite, în ordine, stânga, sus, dreapta și jos. Suprafața depozitului este împărțită în pătrățele de dimensiuni egale, formând linii și coloane. Liniile sunt numerotate de sus în jos cu numerele întregi (1, 2, ...), iar coloanele sunt numerotate de la stânga la dreapta folosind tot numerele întregi (1, 2, ...).

Depozitul conține containere, care păstrează dispozitive cu o valoare inestimabilă. Containerele au numere de identificare distincte. Fiecare container ocupă un pătrățel. Depozitul este suficient de mare pentru ca numărul de containere care intră în depozit să fie mai mic decât numărul de linii, precum și decât numărul de coloane. Containerele nu părăsesc niciodată depozitul dar, uneori, poate sosi un nou container. Intrarea în depozit se află în colțul din stânga sus.

Muncitorul a aranjat containerele în depozit într-un anumit mod, astfel încât să le poată găsi cu ajutorul numărului de identificare. În acest scop el folosește strategia care va fi descrisă în continuare.

Să presupunem că numărul de identifcare al următorului container care sosește este k (îl vom numi containerul k). Muncitorul parcurge mai întâi prima linie, începând din partea stângă și caută primul container care are un număr de identificare mai mare decât k. Dacă un astfel de container nu există, atunci containerul k este plasat în poziția imediat următoare celui mai din dreapta container care se află deja plasat pe acea linie. Dacă găsește un container l pentru care $l > k$, atunci containerul l este înlocuit cu containerul k, iar containerul l este plasat pe linia următoare folosind aceeași metodă. Dacă muncitorul ajunge pe o linie care nu conține nici un container, atunci containerul este plasat în cel mai din stânga pătrățel al acelei linii.

Să presupunem că la intrarea în depozit au ajuns containerele 3, 4, 9, 2, 5 și 1 (în această ordine). În acest caz, rezultă următoarea repartizare a lor în depozit:

```
1 4 5
2 9
3
```

Directorul și muncitorul poartă următorul dialog:
Director: Containerul 5 a sosit înaintea containerului 4?
Muncitor: Nu, este imposibil.
Director: Deci, ești capabil să îmi spui ordinea sosirii containerelor în funcție de poziția lor în depozit.
Muncitor: În general, nu. Dar pentru acest exemplu, containerele care se află acum în depozit ar fi putut sosi în ordinea 3, 2, 1, 4, 9, 5, în ordinea 3, 2, 1, 9, 4, 5 sau în alte 14 moduri posibile.

Deoarece directorul nu vrea să pară mai puțin capabil decât muncitorul, el pleacă. Va trebui să îl ajutați pe director printr-un program care, plecând de la o poziționare a containerelor în depozit, determină toate variantele posibile în care acestea ar fi putut ajunge la depozit.

Intrare

Numele fișierului de intrare este `depot.in`. Prima linie conține un număr întreg R care reprezintă numărul de linii din depozit care conțin containere. Următoarele R linii conțin informații despre containerele de pe cele R linii începând cu latura de sus. Fiecare linie începe cu un număr întreg M care reprezintă numărul containerelor de pe linia respectivă. Urmează M numere întregi care reprezintă numerele de identificare ale containerelor de pe linia respectivă începând din partea stângă. Toate numerele de identificare I ale containerelor satisfac condiția $1 \le I \le 50$. Dacă N este numărul de containere din depozit, atunci $1 \le N \le 13$.

Ieșire

Numele fișierului de ieșire este `depot.out`. Numărul liniilor din fișierul de ieșire este egal cu numărul de posibilități de sosire a containerelor. Fiecare dintre aceste linii conțin N numere întregi care reprezintă numerele de iden-

tificare ale containerelor conform ordinii de sosire care este descrisă de acea linie. Toate liniile descriu ordini de sosire distincte.

Exemple

depot.in	depot.out
3	3 2 1 4 9 5
3 1 4 5	3 2 1 9 4 5
2 2 9	3 4 2 1 9 5
1 3	3 2 4 1 9 5
	3 2 9 1 4 5
	3 9 2 1 4 5
	3 4 2 9 1 5
	3 4 9 2 1 5
	3 2 4 9 1 5
	3 2 9 4 1 5
	3 9 2 4 1 5
	3 4 2 9 5 1
	3 4 9 2 5 1
	3 2 4 9 5 1
	3 2 9 4 5 1
	3 9 2 4 5 1

depot.in	depot.out
2	3 1 2
2 1 2	1 3 2
1 3	

Punctaj

- Dacă fișierul de ieșire nu conține nimic sau conține ordini de sosire imposibile, atunci nu se va acorda nici un punct pentru testul respectiv.
- În caz contrar punctajul este calculat așa cum se descrie în continuare.
  - Dacă fișierul de ieșire conține toate ordinile de sosire posibile o singură dată, atunci veți obține patru puncte.
  - Dacă fișierul de ieșire conține cel puțin jumătate dintre ordinile de sosire posibile o singură dată, atunci veți obține două puncte.
  - Dacă fișierul de ieșire conține mai puțin de jumătate din ordinile de sosire posibile sau unele se repetă, atunci veți obține un punct.

Precizări

- Timpul maxim de execuție admis este de 0.3 secunde/test.
- Limita maximă a memoriei care poate fi folosită este de 32 MB.
- Au fost 25 de teste, pentru rezolvarea fiecăruia acordându-se cel mult 4 puncte.
- Opțiunile de compilare folosite pentru limbajul *Pascal* au fost `-So -O2 -Xs`; opțiunile de compilare folosite pentru limbajele *C* și *C++* au fost `-O2 -static`.

România la CEOI și BOI

Chiar dacă acest număr al revistei este dedicat Olimpiadei Internaționale de Informatică, nu putem să nu amintim și rezultatele obținute de elevii români la celelalte concursuri internaționale care s-au desfășurat în această vară.

Olimpiada de Informatică a Europei Centrale (CEOI) a avut loc în perioada 10-17 august la Zalaegerszeg, Ungaria. Au participat concurenți din 14 țări. Delegația României a fost condusă de lect. univ. Stelian Ciurea (Universitatea "Lucian Blaga" Sibiu) și prof. Doru Popescu Anastasiu (Colegiul Național "Radu Greceanu", Slatina).

La acest concurs au participat elevii clasați pe locurile 5-8 în urma barajelor de selecție. Concurenții români au avut rezultate meritorii:

- Daniel Dumitran (București): medalie de argint (prima după cele de aur...);
- Florin Ghețu (Focșani): medalie de argint;
- Victor Costan (București): medalie de bronz;
- Vencel Bors (Oradea): foarte aproape de medalii.

În paralel s-a desfășurat și "CEOI by Net". Dintre concurenții români menționăm pe Mugurel Andreica (București) și Liviu Lalescu (Craiova), care au obținut medalii de argint.

* * *

Olimpiada Balcanică de Informatică s-a desfășurat între 8 și 13 septembrie în Durres, Albania. Au participat concurenți din 7 țări. Delegația României a fost condusă de lect. univ. Ovidiu Domșa (Universitatea "1 Decembrie 1918" Alba Iulia) și prof. Marinel Șerban (Liceul "Grigore Moisil", Iași).

La acest concurs au participat cei mai buni patru elevi români care nu au terminat încă liceul. Toți au obținut medalii de aur.

- Daniel Dumitran (București) - 580 puncte
- Vencel Bors (Oradea) - 560 puncte;
- Victor Costan (București) - 560 puncte
- Vlad Dascălu (Bacău) - 520 puncte

Concurentul clasat pe a cincia poziție a obținut doar 400 puncte din 600 posibile...

Backtracking în HEAP

Silviu-Gabriel Udrea

În cadrul acestui articol vom prezenta o modalitate de a îmbunătăți timpul de execuție al programelor care rezolvă probleme folosind metoda backtracking. Un truc relativ simplu ne permite să obținem performanțe de aproximativ trei ori mai bune.

Așa cum știm cu toții, una dintre tehnicile de programare care "consumă" mult timp este *backtracking*-ul.

În zilele noastre, când totul se desfășoară cu o viteză incredibilă, o simplă secundă în plus ne pare a fi o infinitate. Înmulțind această "infinitate" cu numărul de teste pe care suntem nevoiți să le efectuăm pentru a ne verifica algoritmii, timpul obținut astfel devine de-a dreptul insuportabil. Și unde anume avem de efectuat cele mai multe calcule, dacă nu la problemele NP-complete, pe care ne vedem obligați să le rezolvăm prin *backtracking*.

Acest motiv ne determină să acordăm o atenție deosebită optimizării de orice fel, căutând în permanență ceva în plus, mai bun.

În clasa a X-a se studiază operațiile cu pointeri, utilizând liste pentru rezolvarea unor anumite tipuri de probleme. Motivele utilizării *heap*-ului sunt numeroase; nu le vom enumera aici, dar cea mai importantă dintre ele este aceea că la nivelul acestuia operațiile se efectuează mult mai rapid. Așa cum am arătat experimental și cum vă invităm și pe dumneavoastră să o faceți, în *heap* operațiile se efectuează aproximativ de 3,3 ori mai rapid față de cum ar fi fost în cazul alocării statice. Totuși, trebuie precizat faptul că acest raport ține de performanțele calculatorului, deci acesta poate varia de la un sistem de calcul la un altul.

Backtracking-ul ne dă întotdeauna dureri de cap. Pentru a le diminua puțin, vom prezenta o metodă generală de a optimiza de cel puțin trei ori timpul de execuție al programelor care folosesc metoda *backtracking*.

Chiar dacă complexitatea problemei, din punctul de vedere al algoritmului rămâne aceeași, programatorul nu va vedea problema cu aceiași ochi, pentru el "complexitatea" rezolvării fiind "optimizată" de trei ori.

Ideea fiind astfel cristalizată, cum rămâne cu implementarea? După cum vom vedea, implementarea nu presupune un efort deosebit.

Ne întrebăm, firesc, care este motivul pentru care am dori să realizăm o astfel de implementare. Vom da un argument pe care considerăm că îl veți găsi suficient de convingător. O metodă pe care probabil o utilizează mulți la rezolvarea problemelor este aceea de a folosi următorul algoritm:

- se rezolvă problema utilizând metoda *backtracking* și se folosește această rezolvare pentru a obține rezultatele corecte pentru date de intrare mici;
- se caută o anumită particularitate;
- dacă s-a găsit o particularitate, atunci aceasta este studiată și se încearcă rezolvarea problemei fără a folosi metoda *backtracking*.

Un exemplu clar al acestei metode ar putea fi prima problemă de la runda a cincia a *Concursului de Programare Agora*. Se rezolvă problema folosind metoda *backtracking*, se observă modul iterativ prin care se putea genera relația de simetrie (soluția era simetrică) și apoi se implementează soluția corectă.

Deoarece algoritmii care folosesc metoda *backtracking* sunt nestandardizați, vom prezenta doar forma generală a *backtracking*-ului în *heap*.

Prezentăm în continuare un tabel cu date obținute experimental la compararea vitezei generării permutărilor de către cele două tehnici, subliniind faptul că rezultatele sunt mediile a mai multor teste (cel puțin 20):

N	7	8	9	10	11
Back	0,05	0,77	8,25	98,17	1268,90
HeapBack	0,00	0,22	2,47	29,70	384,33

Se observă clar că raportul între timpii de execuție ai programului care folosește metoda *backtracking* sunt de aproximativ 3,3 ori mai mari decât cei ai programului care folosește *backtracking în heap*.

Programul care implementează algoritmul poate fi descărcat de la www.ginfo.ro/revista/10_6/main.html.

Silviu-Gabriel Udrea este elev în clasa a XI-a la Colegiul Național "Tudor Vladimirescu" din Târgu-Jiu. Poate fi contactat prin e-mail la adresa silviu_udrea@yahoo.com.

Etapa FINALĂ

A doua ediție a concursului de programare organizat de Gazeta de Informatică s-a încheiat. Primii zece în clasamentul după cele șase runde desfășurate prin e-mail au participat la faza finală care a avut loc pe data de 26 mai 2001 la Liceul de Informatică din Cluj.

Cei zece participanți calificați la faza finală au fost:

Csaba András, Oradea
Mugurel-Ionuț Andreica, București
Ciprian Cană, Vaslui
Adrian Cârcu, Bistrița
George Drumea, București
Liviu Lalescu, Craiova
Maximilian Machedon, București
Cosmin-Silvestru Negruşeri, Bistrița
Mihai Stroe, București
Radu Ștefan, Brașov

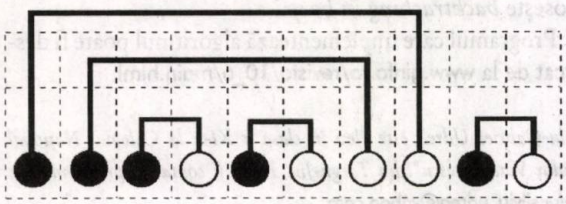
Singurul care nu a putut să se prezinte la faza finală a fost *Csaba András*, așadar am avut doar nouă participanți.

Până la urmă, bursa de 50$/lună oferită de *Computer Press Agora* a fost câștigată de *Mugurel-Ionuț Andreica*. El a fost singurul care a reușit să obțină punctajul maxim la toate cele cinci probleme propuse spre rezolvare. Pe locul al doilea s-a clasat *Mihai Stroe*, care a câștigat o bursă de 30$/lună.

În continuare vă prezentăm enunțurile celor cinci probleme; vă propunem să încercați să le rezolvați. Soluțiile le veți găsi în numărul următor al revistei noastre.

CPAR7P1: Bile

Se consideră n bile negre și n bile albe înșirate pe bară dreaptă. Aceste bile urmează să fie legate între ele două câte două prin alte bare care nu pot să se intersecteze și să se suprapună. În acest scop se vor folosi, în cazul fiecărei perechi, trei bucăți de bare: două verticale și una orizontală, conform figurii. O pereche este formată dintr-o bilă albă și una neagră, sau dintr-una neagră și una albă.



Date de intrare

Fișier de intrare: `BILE.IN`

Linia 1: `n`

- număr natural nenul, reprezentând numărul bilelor albe (și care este totodată și numărul bilelor negre);

Linia 2: `s`

- succesiune de cifre egale cu `0` și `1`, neseparate prin spații, reprezentând bilele fixate pe bară; valoarea `0` corespunde bilelor albe, iar valoarea `1` corespunde bilelor negre.

Date de ieșire

Fișier de ieșire: `BILE.OUT`

Linia 1: `L`

- număr natural nenul, reprezentând lungimea minimă totală a barelor folosite;

Liniile 2..n+1: `x y`

- aceste n linii conțin fiecare, două numere naturale nenule, separate printr-un spațiu, reprezentând numărul de ordine a două bile legate în pereche; perechile care se vor scrie în fișier vor fi ordonate crescător în funcție de valoarea `x`.

Restricții

- $1 \le n \le 100$

Exemplu

BILE.IN	BILE.OUT
5	31
1110100010	1 8
	2 7
	3 4
	5 6
	9 10

Timp maxim de execuție/test: 1 secundă

CPAR7P2: Plăți și încasări

Casierul unei firme trebuie să parcurgă zilnic un traseu liniar prin cele N magazine ale sale. Magazinele sunt numerotate de la 1 la N și se parcurg zilnic în ordinea numerelor

lor de ordine, începând cu primul magazin și terminând cu al N-lea. (Evident casierul va trece doar o singură dată prin fiecare magazin.)

Scopul parcurgerii acestui traseu este efectuarea unor plăți, respectiv a unor încasări. Se știe că există cel puțin un magazin de la care casierul trebuie să încaseze bani. De asemenea se cunosc sumele care trebuie plătite, respectiv încasate.

Într-o zi, patronul nu are bani pentru plăți și îi cere casierului să aleagă două magazine m_i și m_j de pe traseu, astfel încât, parcurgând traseul începând cu m_i și încheindu-l cu m_j, încasările să reprezinte suma maximă posibilă de adus înapoi la firmă.

Casierul trebuie să intre în toate magazinele aflate pe traseul ales, chiar dacă în unele trebuie să efectueze plăți. În plus, pentru a nu crea suspiciuni printre angajații magazinelor, patronul dorește să fie vizitate cât mai multe magazine.

Cerința

Scrieți un program care determină suma maximă posibilă de încasat din magazinele firmei. De asemenea, determinați secvența cea mai lungă de magazine care permite obținerea acestei sume. Dacă aceeași sumă maximă se poate obține pentru mai multe alegeri diferite, vizitând același număr maxim de magazine, atunci în fișierul de ieșire se va scrie una singură, și anume cea care începe de la magazinul cu numărul de ordine cel mai mic.

Date de intrare

Fișier de intrare: `CASIER.IN`

Linia 1: `N`

- număr natural nenul, reprezentând numărul magazinelor de pe traseul inițial.

Linia 2: $\texttt{Suma}_1$ $\texttt{Suma}_2$ ... $\texttt{Suma}_N$

- N numere întregi, separate prin câte un spațiu, reprezentând sumele pe care casierul ar trebui să le plătească sau să le încaseze în/din cele N magazine; numerele pozitive reprezintă sume care se vor încasa, iar numerele negative corespund sumelor care se vor plăti.

Date de ieșire

Fișier de ieșire: `CASIER.OUT`

Linia 1: `SumaMax`

- număr natural nenul, reprezentând suma maximă cu care casierul se întoarce la firmă.

Linia 2: `i j`

- două numere naturale nenule, care aparțin intervalului $[1, N]$ și care reprezintă numerele de ordine ale magazinelor alese de casier pentru începutul și sfârșitul porțiunii de traseu ($i \le j$).

Restricții

- $1 \le N \le 1000$
- $-1000000 \le \texttt{Suma}_i \le 1000000$, $i=1,2,\ldots,N$
- $1 \le \texttt{SumaMax} \le 1000000000$

Exemplu

CASIER.IN	CASIER.OUT
10	13
2 3 -6 5 -6 6 7 -2 2 -1	6 9

Timp maxim de execuție/test: 1 secundă

CPAR7P3: Robot

Un robot se poate mișca în plan doar pe direcțiile verticală, notate cu *up* (`'U'`) și *down* (`'D'`) respectiv orizontală, notate cu *left* (`'L'`) și right (`'R'`). Robotul pornește dintr-un anumit punct și după ce își îndeplinește sarcinile, se întoarce tot în acest punct. Mișcarea robotului se descrie printr-o secvență de comenzi, conform căreia acesta se va deplasa de-a lungul unor linii care formează un caroiaj în care pătrățelele au laturile egale cu 1.

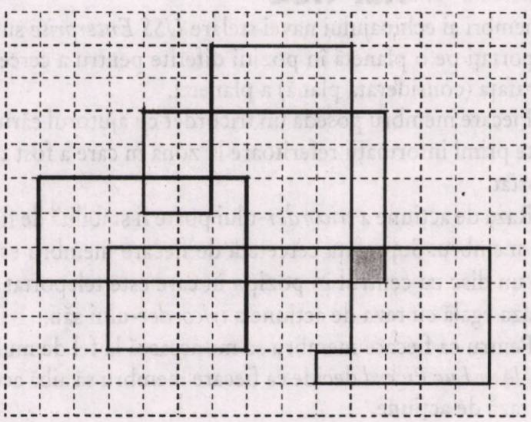
Robotul, în timp ce își parcurge drumul, trasează o linie în urma lui. Datorită faptului că traseul se sfârșește în același punct din care a pornit, se pot forma figuri geometrice, printre care și dreptunghiuri.

Vom numi *dreptunghi vid* acel dreptunghi care se formează din segmente aparținând traseului și care nu este intersectat de alte segmente din traseu.

Cerință

Determinați cel mai mic dreptunghi vid care se formează din traseul robotului.

În imaginea următoare este prezentat un posibil traseu al robotului. Cel mai mic dreptunghi vid care se formează este hașurat.



Date de intrare

Fișier de intrare: `ROBOT.IN`

Linia 1: `x y`

- două numere naturale, separate printr-un spațiu, reprezentând coordonatele punctului de pornire;

Linia 2: `c`

- număr natural nenul, reprezentând numărul de comenzi pe care le execută robotul;

Liniile 3..c+2: *caracter număr*

- unde *caracter* este una din literele `'U'`, `'D'`, `'L'`, `'R'`, având semnificația descrisă mai sus.

Date de ieşire

Fişier de ieşire: `ROBOT.OUT`

Linia 1: c_1 c_2

- două numere naturale, separate printr-un spaţiu, reprezentând coordonatele colţului stânga jos, al dreptunghiului vid având aria minimă.

Linia 2: c_3 c_4

- două numere naturale, separate printr-un spaţiu, reprezentând coordonatele colţului dreapta sus, al dreptunghiului vid având aria minimă.

Restricţie

- coordonatele în care se va afla robotul vor fi întotdeauna în intervalul `[0,100]`.

Exemplu

ROBOT.IN	ROBOT.OUT
4 7	7 3
8	10 5
R 6	
D 4	
L 3	
U 2	
R 5	
D 4	
L 8	
U 6	

Timp maxim de execuţie/test: 1 secundă

CPAR7P4: Star Trek

N membri ai echipajului navei stelare *USS Enterprise* sunt teleportaţi pe o planetă în poziţii diferite pentru a cerceta suprafaţa (considerată plană) a planetei.

Fiecare membru posedă un tricorder cu ajutorul căruia poate primi informaţii referitoare la zona în care a fost teleportat.

Raza de acţiune a *tricorder*-ului poate fi stabilită de fiecare membru. Suprafaţa cercetată de fiecare membru este deci un disc cu centrul în poziţia în care este teleportat şi cu raza egală cu raza de acţiune a *tricorder*-ului său.

Pentru ca fiecare membru să muncească la fel de mult, *Cpt. Jean Luc Picard* decide ca fiecare membru să aibă aceeaşi rază de acţiune.

Pentru a nu apărea interferenţe, nu este permisă intersecţia între suprafeţele studiate de două *tricorder*-e diferite. În plus, raza de acţiune a *tricorder*-elor este obligatoriu un număr natural.

Cerinţă

Stabiliţi raza maximă de acţiune a *tricorder*-elor, astfel încât zonele în care acţionează oricare două dintre ele să nu aibă nici un punct comun.

Date de intrare

Fişier de intrare: `STARTREK.IN`

Linia 1: `N`

- număr natural nenul, reprezentând numărul membrilor echipajului;

Liniile 2..*N*+1: x_i y_i

- pe aceste *N* linii se află câte două numere naturale nenule, separate printr-un spaţiu, reprezentând coordonatele la care sunt teleportaţi membrii echipajului.

Date de ieşire

Fişier de ieşire: `STARTREK.OUT`

Linia 1: `nr`

- număr natural, reprezentând raza maximă de acţiune a *tricorder*-elor.

Restricţii

- $2 \le N \le 10000$
- $0 \le x_i, y_i \le 2000000000$ `i=1,2,...,N`
- Nu pot exista doi membri ai echipajului care să fie teleportaţi în aceeaşi poziţie.

Exemplu

STARTREK.IN	STARTREK.OUT
4	2
0 0	
6 6	
0 6	
6 0	

Timp maxim de execuţie/test: 1 secundă

CPAR7P5: Windows XP

Sistemul de operare *Microsoft Windows XP* va fi lansat oficial pe data de 25 octombrie 2001. După achiziţionarea produsului, mulţi utilizatori vor dori să instaleze noul sistem de operare "pe curat".

Ei vor formata discul hard, vor instala noul sistem de operare, apoi vor dori să instaleze aplicaţiile preferate. După cum ştiţi, instalarea unei aplicaţii necesită, uneori, un anumit număr de resetări ale sistemului. Sarcina voastră este de a-i ajuta pe utilizatori să instaleze toate aplicaţiile, resetând calculatorul de cât mai puţine ori.

Pentru fiecare program de instalare este cunoscut numărul *x* de resetări necesare, precum şi lista aplicaţiilor care trebuie să fie deja instalate pentru ca acesta să poată fi lansat în execuţie.

Un program de instalare funcţionează astfel:

- este lansat în execuţie;
- se execută primul pas al instalării (eventual, în paralel cu execuţia altor programe de instalare);
- se aşteaptă finalizarea execuţiei pasului curent de către toate programele de instalare aflate în execuţie;
- se resetează sistemul ;
- se trece la al doilea pas, etc.

La un moment dat se pot afla în execuţie oricâte programe de instalare, dacă aplicaţiile necesare pentru execuţia lor au fost deja instalate.

Să presupunem că dorim să instalăm aplicațiile *Microsoft Office XP*, *Microsoft Visual Studio.NET*, *Microsoft DirectX 8.0* și *Microsoft Internet Explorer 6.0*. Pentru a instala *Office* este nevoie de patru resetări, pentru *Visual Studio* este nevoie de trei resetări, pentru *DirectX* două resetări și pentru *Internet Explorer* o resetare. *Internet Explorer* și *DirectX* pot fi instalate fără să fie instalată nici o altă aplicație. *Visual Studio* poate fi instalat numai dacă sunt instalate *DirectX* și *Internet Explorer*, iar *Office* dacă a fost instalat *Internet Explorer*.

La primul pas se vor lansa programele de instalare pentru *Internet Explorer* și *DirectX*. După resetare, *Internet Explorer* este instalat, iar pentru *DirectX* se trece la al doilea pas al instalării.

Datorită faptului că *Internet Explorer* este instalat, se poate lansa în execuție și programul de instalare pentru *Office*. La acest al doilea pas se află în execuție programele de instalare pentru *DirectX* (al doilea pas) și *Office* (primul pas).

După resetare, *DirectX* este instalat, iar pentru *Office* se trece la al doilea pas. Datorită faptului că *DirectX* este instalat, se poate lansa în execuție și programul de instalare pentru *Visual Studio*. La acest al treilea pas se află în execuție programele de instalare pentru *Office* (al doilea pas) și *Visual Studio* (primul pas).

După resetare se vor afla în execuție programele de instalare pentru *Office* (al treilea pas) și *Visual Studio* (al doilea pas). După o nouă resetare se vor afla în execuție programele de instalare pentru *Office* (al patrulea pas) și *Visual Studio* (al treilea pas).

Va avea loc o ultimă resetare, după care și aceste două programe sunt instalate. În total au avut loc cinci resetări.

Date de intrare

Fișier de intrare: `WINXP.IN`

Linia 1: `N`

- număr natural nenul, reprezentând numărul aplicațiilor care trebuie instalate;

Liniile 2..*N*+1: `nr na` a_1 $a_2 \ldots a_{na}$

- aceste *N* linii conțin fiecare mai multe numere naturale, separate prin câte un spațiu; *nr* reprezintă numărul de resetări pentru aplicația respectivă, *na* reprezintă numărul de aplicații care trebuie deja să fie instalate, iar a_1, a_2, ..., a_{na} reprezintă numerele de ordine ale aplicațiilor care trebuie să fie deja instalate (dacă este cazul).

Date de ieșire

Fișier de ieșire: `WINXP.OUT`

Linia 1: `R`

- număr întreg pozitiv, reprezentând numărul total de resetări;

Liniile 2..*R*+1: nr_1 nr_2 ...

- aceste *R* linii conțin fiecare un număr oarecare de numere naturale nenule, reprezentând numerele de ordine ale aplicațiilor a căror programe de instalare se află în execuție la pasul respectiv.

Exemplu

WINXP.IN	WINXP.OUT
4	5
4 1 4	3 4
3 2 3 4	1 4
2 0	1 2
1 0	1 2
	1 2

Timp maxim de execuție/test: 1 secundă

Despre concurs

Concurenții au considerat că toate cele cinci probleme au fost relativ simple, dovadă fiind faptul că au fost obținute multe punctaje de 100 de puncte la diferite probleme. Totuși, doar câștigătorul concursului le-a rezolvat corect pe toate cinci.

Problema ***Bile*** a fost rezolvată corect de *Mugurel-Ionuț Andreica*, *Adrian Cârcu* și *Mihai Stroe*.

La problema ***Plăți și încasări*** au fost obținute punctaje maxime de către *Mugurel-Ionuț Andreica*, *Ciprian Cană*, *Adrian Cârcu*, *Cosmin-Silvestru Negrușeri*, *George Drumea*, *Maximilian Machedon* și *Ștefan Radu*.

A treia problemă, ***Robot***, a fost rezolvată corect de doar doi concurenți: *Mugurel-Ionuț Andreica* și *Ștefan Radu*.

Tot doi concurenți, *Mugurel-Ionuț Andreica* și *Mihai Stroe*, au obținut punctaje maxime la problema ***Star Trek***.

În fine, problema ***Windows XP*** a fost rezolvată corect de *Mugurel-Ionuț Andreica*, *Adrian Cârcu*, *Cosmin-Silvestru Negrușeri*, *George Drumea*, *Ștefan Radu* și *Mihai Stroe*.

Programele celor nouă participanți au fost evaluate folosind un calculator dotat cu procesor *Intel Celeron 600*, 56 MB de memorie internă și folosind sistemul de operare *Microsoft Windows '98 Second Edition*. Limitele de timp indicate în revistă sunt valabile pentru acest tip de calculator.

Diplomele primite de concurenți au fost înmânate în prezența domnului ***Cristian Nagy***, reprezentatul editurii ***Computer Press Agora***.

Toți participanții au primit din partea organizatorilor câte un abonament pe un an la *GInfo*.

Concluzii

Această ediție a ***Concursului de Programare Agora*** s-a dovedit a fi un real succes, dovadă fiind numărul mare al celor înscriși (peste 400, dintre care aproape 100 din afara granițelor țării).

Îi felicităm pe toți cei care au participat și le mulțumim pentru interesul acordat concursului nostru. Le urăm tuturor mult succes la ediția 2001/2002 a concursului și sperăm că aceasta va fi cel puțin la fel de interesantă și îi va ajuta la fel de mult pe participanți la pregătirea pentru diferite concursuri naționale și internaționale de programare.

Bursele AGORA

Mulți dintre cititorii revistei noastre au dorit să câștige una dintre bursele oferite de editura Agora Media. Câștigatorii ultimei ediții au fost deja desemnați, dar acum începe un nou concurs. Vă dorim succes!

Despre concurs

Pentru a participa la ediția din acest an a concursului nostru vă rugăm să citiți cu atenție noul regulament. Există câteva modificări minore care constau, în special, în modalitatea de trimitere a soluțiilor și cea de înscriere.

Pentru a vă înscrie vă rugăm să completați integral talonul de participare publicat în revistă. Taloanele completate incomplet nu vor fi luate în considerare. Redacția *GInfo* nu își asumă nici o răspundere pentru taloanele care, din diferite motive, nu ajung în timp util la redacție.

Prima rundă

La prima rundă va trebui să rezolvați trei probleme: *Matrice*, *Dune* și *Warcraft*. Prima dintre ele a fost propusă de *Mihai Stroe*, student în anul IV la *Universitatea Politehnica* din București, fost olimpic internațional, câștigător al primei ediții a concursului nostru și clasat pe locul al doilea la cea de-a doua. Celelalte probleme au fost propuse de către *Mihai Scorțaru*, student în anul IV la *Universitatea Tehnică* din Cluj, redactor-șef adjunct al revistei noastre și organizator principal al concursului.

Pentru a putea participa la prima rundă va trebui să trimiteți talonul din acest număr al revistei până pe data de ***15 noiembrie 2001*** (data poștei). În termen de ***cinci*** zile de la primirea talonului de către redacție, vă vom trimite prin e-mail codul dumneavostră de identificare. Rezolvările celor trei probleme pot fi trimise până pe data de ***30 noiembrie 2001***. Rezultatele primei runde vor deveni publice pe data de ***15 decembrie 2001***.

Site-ul concursului

Am creat și un *site* special dedicat concursului. Adresa acestuia este **http://www.ginfo.ro/concurs/main.html**.

Pe *site* veți putea găsi enunțurile problemelor, soluțiile oficiale, anunțurile generale, precum și punctajele obținute de concurenți.

Enunțurile problemelor vor fi publicate în prima zi a fiecărei luni începând cu luna noiembrie. Astfel, problemele primei runde vor fi publicate pe data de 1 noiembrie, problemele celei de-a doua runde pe data de 1 decembrie etc.

Soluțiile oficiale vor apărea pe *site* tot în prima zi a fiecărei luni începând cu luna decembrie. Vor fi publicate și datele de test folosite pentru evaluare, precum și datele de ieșire corecte. În cazul în care există mai multe ieșiri corecte, numai una va fi publicată. Astfel, soluțiile problemelor de la prima rundă vor fi publicate pe data de 1 decembrie, soluțiile problemelor de la cea de-a doua runde pe data de 1 ianuarie etc.

Rezultatele concurenților pentru fiecare rundă vor fi publicate pe data de 15 a lunii începând cu luna decembrie. Astfel, rezultatele primei runde vor fi publicate pe data de 15 decembrie, rezultatele celei de-a doua runde pe data de 15 ianuarie etc.

Pe *site* vor fi publicate și eventualele anunțuri generale referitoare la probleme.

Restricții și precizări

Nu se vor accepta rezolvări care ajung la redacție după termenul limită specificat, chiar dacă acesta este depășit doar cu câteva minute.

Redacția nu va trimite prin e-mail enunțurile problemelor, indiferent de motivul pentru care un concurent ar putea solicita acest lucru.

Termenul limită de trimitere a problemelor este ultima zi a lunii (până la ora 24:00).

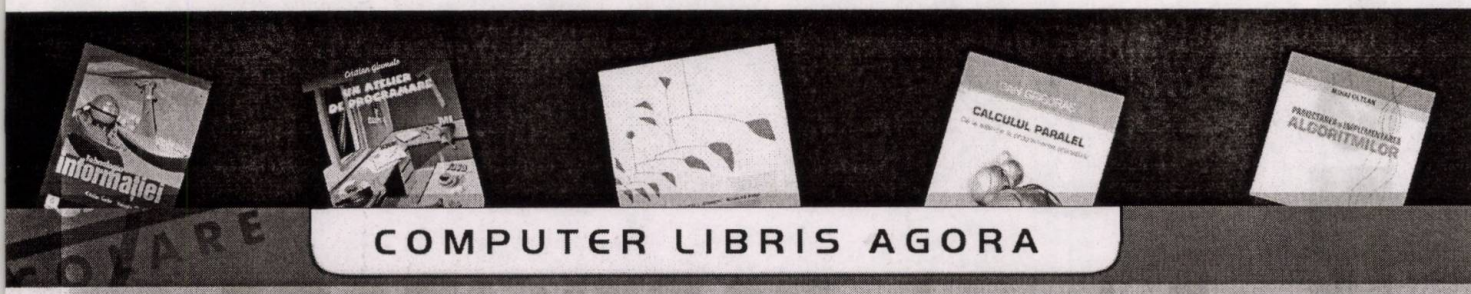
Organizatorii concursului își rezervă dreptul de a modifica termenele de publicare a problemelor, a soluțiilor sau a rezultatelor și a termenelor de trimitere a soluțiilor.

De asemenea, unele prevederi ale regulamentului pot fi eliminate, modificate sau pot fi adăugate altele noi. Acestea vor fi publicate în paginile revistei și vor intra în vigoare începând cu runda pentru care au fost publicate enunțurile problemelor în același număr al revistei.

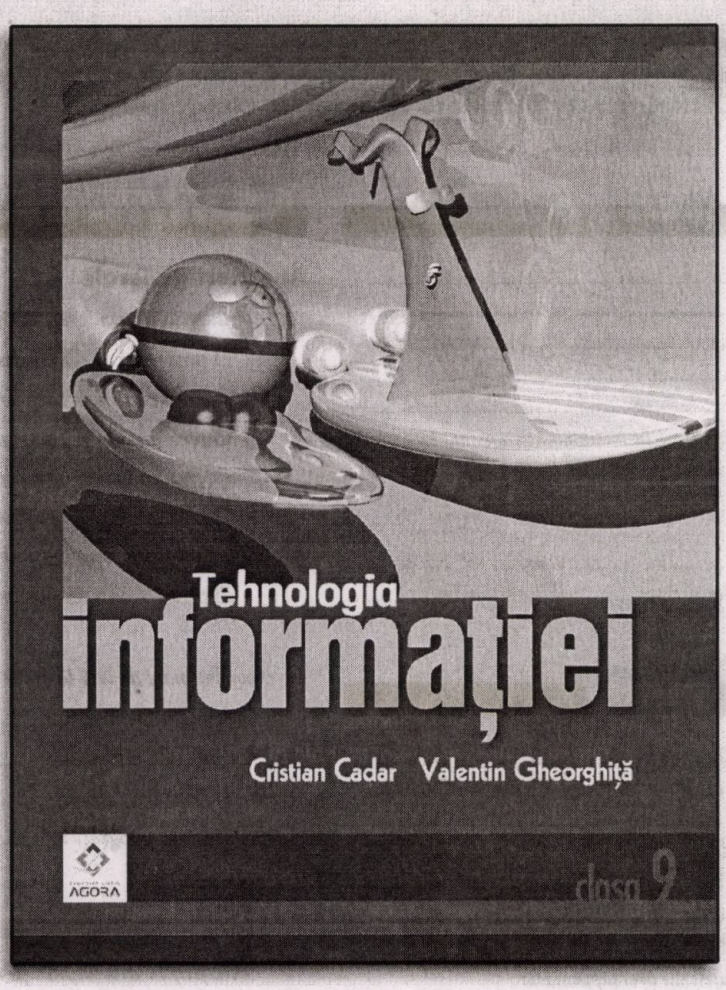
Pentru eventualele contestații, vă rugăm să folosiți adresa e-mail a concursului. Vă recomandăm ca înainte de trimiterea unei contestații să verificați punctajul obținut folosind datele de test care sunt publicate pe *site*.

Pentru detalii suplimentare referitoare la concurs, vă rugăm să contactați membrii redacției revistei noastre.

Mult succes tuturor participanților!!!



COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS
AGORA

Rundele 1, 2, 3, 4, 5!

Talon de PARTICIPARE

Acest talon dă dreptul de participare la Concursul de Programare Agora începând cu prima rundă. Completați integral acest talon și trimiteți-l pe adresa redacției: str. Universității nr. 8 ap. 8, Cluj-Napoca.

Date personale

Nume: .....
Prenume: .....
Telefon: .....
Adresă e-mail: .....
❒ *elev în clasa* ..... *la* .....
❒ *student în anul* ..... *la* .....
Data nașterii: .....

Adresa

str. ..... *nr.* .... *bl.* ..... *sc.* .....
apt. ..... *cod* ..... *localitate* ..... *județ* .....

Chestionar

Folosiți un calculator acasă? ❒ *Da* ❒ *Nu*
Aveți acces e-mail acasă? ❒ *Da* ❒ *Nu*
Aveți acces WWW acasă? ❒ *Da* ❒ *Nu*
Plănuiți un upgrade anul acesta? ❒ *Da* ❒ *Nu*
Cât timp petreceți în fața calculatorului? ..... *ore/săptămână*
Cât timp petreceți online? ..... *ore/săptămână*
Cât timp vă jucați? ..... *ore/săptămână*
Cât timp programați? ..... *ore/săptămână*
Câte calculatoare sunt la școală/facultate? ..... *buc.*
Cum apreciați dotarea cu tehnică de calcul a școlii?
❒ *Dezastruoasă* ❒ *Foarte slabă* ❒ *Slabă*
❒ *Nesatisfăcătoare* ❒ *Medie* ❒ *Satisfăcătoare*
❒ *Bună* ❒ *Foarte bună* ❒ *Excelentă*
Considerați că prețul revistei este:
❒ *prea mic* ❒ *accesibil* ❒ *prea mare*
Ați dori ca revista să conțină și un CD? ❒ *Da* ❒ *Nu*
Ați dori ca revista să fie color? ❒ *Da* ❒ *Nu*
De cât timp cunoașteți GInfo? ..... *ani*
Cum ați procurat acest număr?
❒ *chioșc* ❒ *abonament* ❒ *cadou* ❒ *altfel:* .....
Ați dori ca GInfo să aibă:
❒ *4 numere/an* ❒ *8 numere/an*
❒ *12 numere/an* ❒ *24 numere/an*

Conținutul GInfo

Aprecieri generale

Considerați că ponderea problemelor rezolvate este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că ponderea problemelor propuse este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că ponderea subiectelor teoretice este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că nivelul de accesibilitate al articolelor este:
❒ *prea coborât* ❒ *potrivit* ❒ *prea ridicat*

Acordați note de la 1 la 10 pe următoarele criterii:
Structura tematică: .....
Calitatea articolelor: .....
Prezentare grafică: .....
Calitate tipografică: .....

În viitor

În următoarele numere așteptați:
mai multe articole de programare? ❒ *Da* ❒ *Nu*
mai multe articole teoretice? ❒ *Da* ❒ *Nu*
mai multe articole despre Internet? ❒ *Da* ❒ *Nu*
mai multe prezentări de programe? ❒ *Da* ❒ *Nu*
prezentări de jocuri? ❒ *Da* ❒ *Nu*
articole despre hardware? ❒ *Da* ❒ *Nu*

V-ar plăcea să citiți:
pagini redactate de elevii unor licee? ❒ *Da* ❒ *Nu*
lucrări premiate, realizate de elevi? ❒ *Da* ❒ *Nu*
actualități școlare? ❒ *Da* ❒ *Nu*
prezentarea unor situri Web? ❒ *Da* ❒ *Nu*
tips & tricks? ❒ *Da* ❒ *Nu*

Calculatorul

Indicați caracteristicile calculatorului pe care îl veți folosi la rezolvarea problemelor de la concurs.

Tipul procesorului

- ❒ *Intel Pentium*
- ❒ *Intel Pentium Pro*
- ❒ *Intel Pentium II*
- ❒ *Intel Pentium III*
- ❒ *Intel Pentium 4*
- ❒ *Intel Celeron*
- ❒ *AMD K6*
- ❒ *AMD K6-2*
- ❒ *AMD K6-III*
- ❒ *AMD Athlon*
- ❒ *AMD Duron*
- ❒ *Altul:* .....

***Frecvența procesorului*:** ..... MHz

***Tipul plăcii de bază*:** .....

***Frecvența magistralei sistem*:** ..... MHz

***Capacitatea memoriei sistem*:** ..... MB

***Tipul plăcii video*:** .....

***Capacitatea memoriei plăcii video*:** ..... MB

***Tipul discului hard*:** .....

***Capacitatea discului hard*:** ..... GB

Componente suplimentare

Unitate CD-ROM: ❒ Da ❒ Nu
Tip: .....
Viteza de citire: ..... ×

Unitate CD-RW: ❒ Da ❒ Nu
Tip: .....
Viteza de citire: ..... ×
Viteza de inscripționare: ..... ×
Viteza de reinscripționare: ..... ×

Placă de sunet: ❒ Da ❒ Nu
Tip: .....

Boxe active: ❒ Da ❒ Nu
Tip: .....

Placă de rețea: ❒ Da ❒ Nu
Tip: .....

Modem: ❒ Da ❒ Nu
Tip: .....
Viteza de transfer: ..... Kbps

Sistemul de operare

- ❒ *MS-DOS*
- ❒ *Microsoft Windows '95 OSR 2*
- ❒ *Microsoft Windows '98 Second Edition*
- ❒ *Microsoft Windows NT 4.0*
- ❒ *Microsoft Windows XP*
- ❒ *Microsoft Windows '95*
- ❒ *Microsoft Windows '98*
- ❒ *Microsoft Windows Me (Millennium Edition)*
- ❒ *Microsoft Windows 2000*
- ❒ *Altul:* .....

Aplicații folosite frecvent

Marcați aplicațiile instalate pe calculator și indicați versiunile corespunzătoare

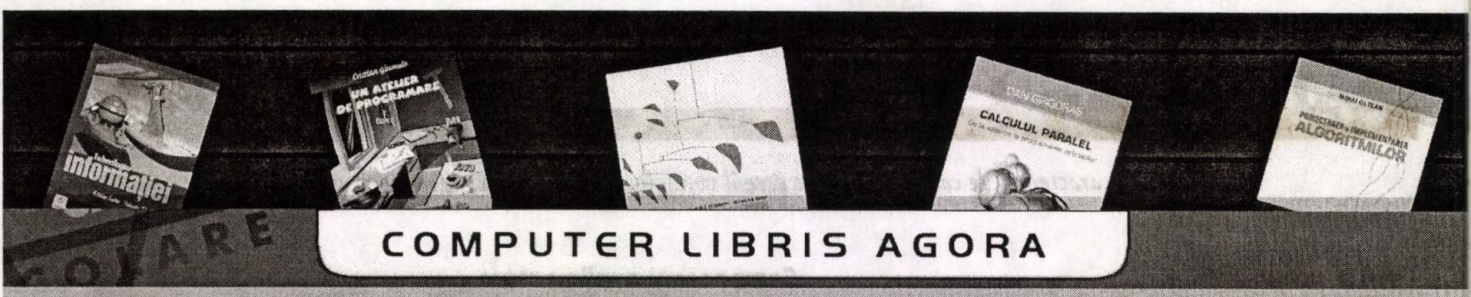
- ❒ *Microsoft Internet Explorer (versiunea .....)*
- ❒ *Microsoft Office (versiunea .....)*
- ❒ *Microsoft Visual Studio (versiunea .....)*
- ❒ *Borland Pascal (versiunea .....)*
- ❒ *Corel Draw (versiunea .....)*
- ❒ *Borland C++Builder (versiunea .....)*
- ❒ *Netscape Navigator (versiunea .....)*
- ❒ *Adobe Photoshop (versiunea .....)*
- ❒ *Adobe Acrobat Reader (versiunea .....)*
- ❒ *Borland C++ (versiunea .....)*
- ❒ *Borland Delphi (versiunea .....)*
- ❒ *Borland JBuilder (versiunea .....)*

***Programe antivirus utilizate*:** .....

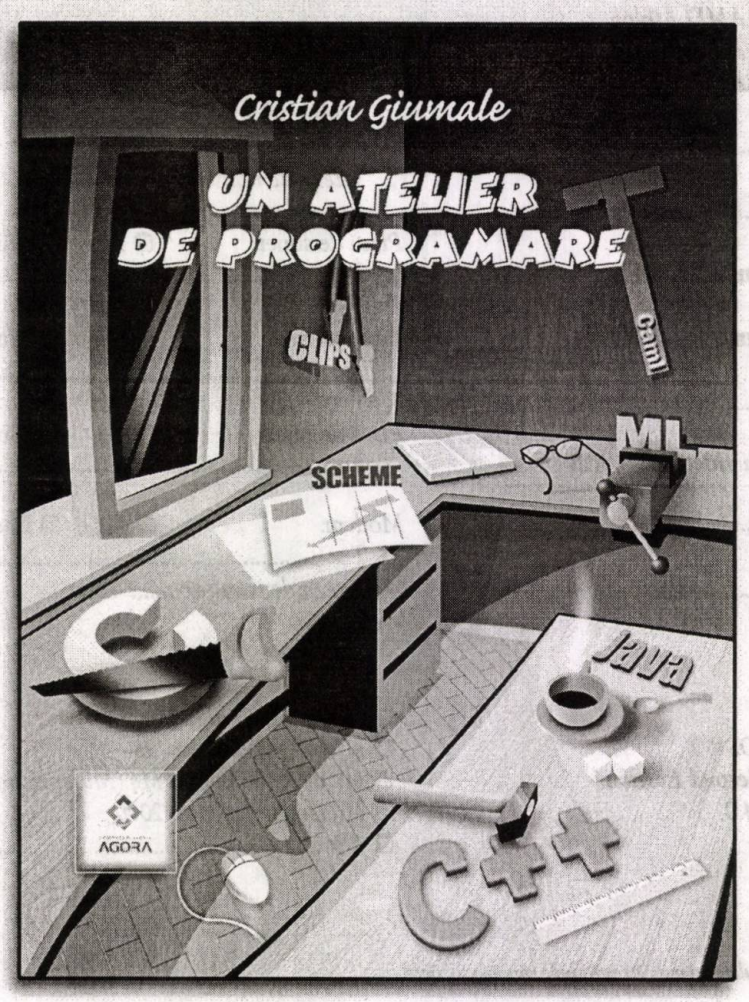
***Arhivatoare utilizate*:** .....

***Jocurile preferate*:** .....

***Site-urile preferate*:** .....



COMPUTER LIBRIS AGORA



Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

CONCURS de programare

Ediția a treia a concursului de programare organizat de Gazeta de Informatică începe în această lună. Pentru detalii referitoare la acest concurs citiți paginile următoare sau vizitați site-ul nostru: www.ginfo.ro.

Informații generale

Concursul *Bursele Agora* este o competiție internațională destinată elevilor și studenților. Acest concurs este organizat de redacția *GInfo* și editura *Agora Media*.

Vor fi cinci runde, la fiecare dintre ele participanții trebuie să rezolve trei probleme și să trimită soluțiile prin e-mail; adresa de e-mail a concursului este concurs@ginfo.ro. După cele cinci etape, care se vor desfășura prin e-mail, va fi o etapă finală care se va desfășura la *Cluj-Napoca*; doar primii zece clasați în urma celor cinci runde vor participa la faza finală.

Problemele care trebuie rezolvate sunt propuse de redacția *GInfo* în colaborare cu studenți și profesori care se ocupă de pregătirea elevilor români care participă la diferite concursuri naționale și internaționale.

Înscrierea

Pentru a participa la concursul *Bursele Agora* va trebui mai întâi să vă înscrieți. Pentru înscriere este necesară completarea formularului din revistă și trimiterea acestuia la adresa:

Redacția GInfo
str. Universității nr. 8/8
3400 Cluj-Napoca

Vă vom trimite un e-mail care va conține codul dumneavoastră de identificare (ID).

Regulament

Limbajele de programare acceptate la *AM* sunt *Pascal* și *C/C++*. Fișierele sursă vor fi compilate folosind *Borland Pascal 7.0* sau *Borland C++ 3.1*.

Problemele vor apărea în paginile revistei *GInfo* și pe site-ul *Web*, iar rezolvările vor trebui trimise înaintea termenului limită specificat pentru fiecare rundă.

O singură soluție va fi acceptată pentru fiecare problemă. Dacă un concurent va trimite mai multe soluții pentru o anumită problemă numai prima soluție va fi luată în considerare, restul fiind ignorate.

Redacția *GInfo* își rezervă dreptul de a descalifica concurenții care trimit soluții identice (sau aproape identice).

Este interzisă folosirea de secvențe scrise în limbaj de asamblare; dacă un fișier sursă conține astfel de secvențe concurentul va primi 0 puncte pentru problema respectivă.

Este interzis ca programele generate folosind sursele concurenților să modifice *timer*-ul. Concurenții care vor încerca să modifice ceasul sistem vor fi descalificați.

Dacă aveți neclarități referitoare la enunțurile problemelor, puteți pune întrebări folosind adresa de e-mail a concursului.

Nu puteți pune decât întrebări ale căror răspunsuri posibile sunt `DA` sau `NU`. Dacă răspunsul la întrebare pote fi găsit în enunțul problemei, atunci răspunsul la întrebare va fi `FĂRĂ COMENTARII`.

Dacă redacția *GInfo* consideră că răspunsul la o anumită întrebare trebuie primit de toți participanții atunci cel care a adresat întrebarea va primi răspunsul `ANUNT GENERAL`. Acest răspuns va fi urmat de un e-mail pe care îl vor primi toți participanții și care va clarifica nelămurirea.

Trimiterea soluțiilor

Pentru a trimite soluțiile de la o anumită rundă va trebui să trimiteți un e-mail având ca subiect codul dumneavoastră de identificare urmat de textul `Runda X`, unde `X` reprezintă numărul rundei (1-5). Acest e-mail va avea atașat o arhivă `.zip` sau `.rar` având numele `yyyyRx.ext` unde `yyyy` este codul dumnevoastră de identificare, `x` este numărul rundei, iar `ext` este extensia (`RAR` sau `ZIP`).

Arhiva va conține un singur director numit `yyyy` (codul dumneavoastră de înregistrare). Acest director trebuie să conțină trei subdirectoare, corespunzătoare celor trei probleme ale rundei și un fișier `.BAT` care va conține liniile de comandă pe care doriți să le folosim pentru a vă compila sursele. Denumirile subdirectoarelor sunt date în descrierea problemelor, iar a fișierului va fi `ROUNDX.BAT`. Fiecare dintre cele trei subdirectoare va conține fișierul sursă corespunzător problemei respective. Denumirea acestuia va fi `yyyy.pas`, `yyyy.c` sau `yyyy.cpp`. Din motive de securitate arhivele autoexpandabile nu sunt acceptate.

Arhiva nu poate conține nici un alt fișier sau director, deci nu puteți trimite fișiere auxiliare împreună cu fișierele

sursă. Dacă programul dumneavoastră trebuie să creeze fișiere auxiliare atunci acestea vor avea extensia `.TMP` și trebuie șterse înaintea terminării execuției programului. Dacă această regulă nu este respectată concurentul va primi 0 puncte pentru problema corespunzătoare chiar dacă soluția este corectă. Dacă redacția *GInfo* consideră că un concurent pune în pericol (prin suprascrierea unor fișiere, crearea de fișiere foarte mari etc.) buna desfășurare a competiției atunci acel concurent poate fi descalificat.

Toate fișierele trimise devin proprietatea editurii *Agora Media* și nu pot fi folosite în scopuri comerciale fără acordul scris al editurii.

Evaluarea

Pentru fiecare problemă puteți primi maximum 100 de puncte; numărul maxim de puncte pentru o anumită rundă este 300, iar pentru întreg concursul 1500. Pentru corectarea fiecărei probleme vom folosi între 5 și 20 seturi de date; dacă programul va genera un fișier de ieșire corect pentru un anumit set de date atunci concurentul va primi un număr de puncte care se va adăuga la numărul de puncte corespunzător problemei respective.

Calculatorul folosit pentru evaluare are următoarea configurație: procesor *Intel Pentium™ III 500 MHz*, RAM: 128 MB, HDD: 13 GB. Este garantat faptul că în timpul evaluării vor fi liberi cel puțin 400 KB de memorie convențională și cel puțin 100 MB de spațiu pe disc. Sistemul de operare folosit în timpul evaluării va fi *Microsoft Windows XP Home Edition*. Dacă un program va efectua o operație ilegală atunci concurentul va primi 0 puncte pentru setul de date corespunzător.

În cazul în care configurația hardware sau software a calculatorului folosit pentru evaluare se modifică, toți concurenții vor fi înștiințați prin e-mail.

Sfaturi

Programatorii în *Pascal* nu ar trebui să folosească niciodată unitul `CRT` deoarece programele lor se vor opri returnând eroarea de execuție `Error 200: Division by zero`. Acest fapt este cauzat de un binecunoscut bug al mediului de programare *Borland Pascal 7.0*; nici o problemă propusă spre rezolvare în cadrul concursului *Bursele Agora* nu necesită folosirea acestui unit.

Atunci când sunt folosite anumite tipuri de date (cum ar fi `Comp`) modul `8087` trebuie să fie activ. Opțiunile de mediu permit setarea acestui mod ca mod implicit, dar în mod normal acest mod nu este activ. Programatorii în *Pascal* sunt rugați să includă în codul sursă directiva de compilare `{$N+}` dacă doresc să folosească tipuri de date care necesită activarea acestui mod. În caz contrar, în mometul în care vom încerca să compilăm fișierul sursă, va fi generată eroarea de compilare `Error 116: Must be in 8087 mode to compile this`. Pentru a vă asigura că programul se va compila exact ca pe calculatorul folosit pentru a-l scrie, apăsați `CTRL+O,O`. Astfel, în sursa dumneavoastră, vor fi incluse toate directivele de compilare cu opțiunile care sunt setate în mediu. Puteți modifica aceste opțiuni astfel încât să fie cele dorite de dumneavoastră. Nu vă îngrijorați dacă nu cunoașteți semnificația unor anumite directive de compilare; acestea nu vor avea efecte negative asupra execuției programului - dacă programul dumneavoastră va rula corect pe calculatorul pe care l-ați folosit pentru a-l scrie, atunci va rula corect și pe calculatorul folosit pentru evaluare.

Toate programele trebuie să citească datele de intrare din fișierul de intrare specificat. Nu ar trebui să fie așteptate date de la intrarea standard (tastatură). Dacă programul dumneavoastră va aștepta o reacție de la tastatură atunci execuția sa va fi oprită după expirarea timpului și nu veți obține nici un punct pentru acea soluție chiar dacă ea este corectă. Deci, vă rugăm să nu includeți în codul sursă secvențe ca `Readln`, `getch()`, `getche()` etc.

Toate programele trebuie să scrie datele de ieșire în fișierul de ieșire specificat. Puteți scrie date și la ieșirea standard, dar aceasta nu va duce decât la încetinirea programului, deci nu este recomandabil să scrieți date pe ecran. Toate datele scrise la ieșirea standard sunt ignorate în timpul evaluării. Programul de evaluare va verifica numai soluția scrisă în fișierul de ieșire specificat.

Fișierul de intrare se află întotdeauna în directorul curent; nu încercați să citiți date din alte locații. Fișierul de ieșire va trebui generat tot în directorul curent; nu scrieți date de ieșire în alte locații. Trebuie să folosiți întotdeauna denumirile specificate în enunț pentru fișierul sursă, fișierul de intrare și fișierul de ieșire. Folosirea altor denumiri va duce la funcționarea incorectă a programului.

Nu trimiteți soluții după termenul limită (indicat în fiecare număr al revistei și pe site). Nici o astfel de soluție nu va fi evaluată!!!

Exemplu

La fiecare rundă vor fi trei probleme care trebuie rezolvate. Să presupunem că una dintre ele are următorul enunț:

Nume subdirector: **SUMA**
Nume fișier de intrare: **SUMA.IN**
Nume fișier de ieșire: **SUMA.OUT**
Descriere: Dându-se două numere, aflați suma lor.
Date de intrare: Fișierul de intrare conține, pe o singură linie, cele două numere separate printr-un singur spațiu.
Date de intrare: Fișierul de ieșire ar trebui să conțină suma celor două numere.
Exemplu:

SUMA.IN	SUMA.OUT
2 4	6

Timp maxim de execuție: 1 secundă/test

După citirea enunțului veți începe să rezolvați problema. Veți observa că nu este clar dacă numerele sunt întregi sau reale.

Pentru a clarifica acest lucru va trebui să adresați o întrebare. Aceasta nu poate fi: *Numerele sunt întregi sau reale?* deoarece nu se poate răspunde la ea prin `DA` sau `NU`. O

soluție ar fi să întrebați: *Numerele sunt întregi?* Veți primi răspunsul NU și veți ști că numerele sunt reale.

Acum apare o întrebare mult mai importantă: *Ce precizie trebuie folosită pentru afișarea rezultatului?* Se pare că redacția *GInfo* a uitat să precizeze acest lucru în enunț. Bineînțeles, va trebui să adresați o întrebare la care să se poată răspunde prin DA sau NU. Ați putea încerca: *Este suficientă o precizie de trei zecimale?* Răspunsul la întrebare va fi ANUNT GENERAL și va fi urmat de un alt e-mail care va conține un mesaj de genul:

Rezultatul va trebui scris cu o precizie de două zecimale. Ne cerem scuze pentru descrierea incompletă a problemei.

Sperăm să nu fim niciodată nevoiți să trimitem astfel de mesaje, dar... După primirea anunțului general puteți începe scrierea codului sursă. O soluție corectă ar putea fi:

```
 1: Program SUMA;
 2: var a,b:Real;
 3:     f:Text;
 4: Begin
 5:   Assign(f,'SUMA.IN');
 6:   Reset(f);
 7:   Readln(f,a,b);
 8:   Close(f);
 9:   Assign(f,'SUMA.OUT');
10:   Rewrite(f);
11:   Writeln(f,a+b:0:2);
12:   Close(f)
13: End.
```

Numerele de la începutul fiecărei linii nu fac parte din program; ele sunt folosite pentru a ne putea referi mai ușor la o anumită linie.

Dacă veți adăuga o linie `1': uses Crt;` și o linie `4': Clrscr;` nu veți reuși să obțineți decât eroarea `Error 200: Division by zero` așa cum am explicat în secțiunea *Sfaturi*.

Dacă adăugați o linie `12': Readln;` rezultatul evaluării va fi `Time out!` deoarece execuția programului nu se va termina, ci se va aștepta apăsarea tastei <ENTER>. Execuția va fi întreruptă de către programul de evaluare după o secundă.

Dacă folosiți linia `5: Assign(f,'C:\SUMA.IN');` atunci programul va funcționa corect numai dacă directorul curent este rădăcina drive-ului `C:` (ceea ce este foarte improbabil). Nu veți obține decât eroarea `File not found`.

Dacă folosiți linia `5: Assign(f,'ALTFISIER');` atunci programul nu va citi datele de intrare din fișierul specificat și va rula incorect.

Dacă folosiți linia `9: Assign(f,'C:\SUMA.OUT');` atunci programul va părea că funcționează corect dar programul de evaluare nu va găsi fișierul dumneavostră de ieșire decât dacă directorul curent este rădăcina drive-ului `C:` (din nou, șansele sunt foarte mici). Programul de evaluare va returna `No output file!`.

Dacă folosiți linia `9: Assign(f,'ALTFISIER');` atunci programul nu va scrie datele în fișierul de ieșire specificat, iar programul de evaluare va returna `No output file!`.

Dacă folosiți linia `7: Readln(a,b);` atunci programul va aștepta datele de intrare de la intrarea standard și va fi întrerupt după o secundă. Puteți folosi această linie numai dacă aveți o secvență asemănătoare cu:

```
5: Assign(Input,'SUMA.IN');
6: Reset(Input);
7: Readln(a,b);
8: Close(Input);
```

Dacă folosiți linia `11: Writeln(a+b:0:2);` atunci datele de ieșire vor fi scrise pe ecran și nu în fișierul de ieșire specificat. Puteți folosi această linie numai dacă secvența este de tipul:

```
 9: Assign(Output,'SUMA.OUT');
10: Rewrite(Output);
11: Writeln(a+b:0:2);
12: Close(Output)
```

Acestea sunt cele mai des întâlnite erori care au apărut la prima ediție a concursului organizat de *GInfo*. Sperăm că nu veți face aceleași greșeli și la această ediție.

După scrierea codului sursă trebuie să trimiteți soluția împreună cu soluțiile celorlalte două probleme. Să presupunem că denumirile directoarelor pentru aceste probleme sunt **DIF** și **PRODUS**. Trebuie să creați arhiva pe care o veți trimite. Această arhivă va conține un director numit yyyy unde yyyy este numărul dumneavostră de identificare (de exemplu, 2457). Veți crea directorul **2457** și veți crea în interiorul său trei subdirectoare numite **SUMA**, **DIF** și **PRODUS**. Fiecare dintre cele trei subdirectoare va conține fișierul sursă corespunzător problemei respective. Fiecare dintre cele trei fișiere va avea numele **2457.EXT** (unde **EXT** este extensia corespunzătoare limbajului ales și poate fi **PAS**, **C** sau **CPP**). Acum trebuie creat fișierul **.BAT** pe care îl vom folosi pentru compilarea surselor. Aceasta se va numi **RUNDA1.BAT** și ar putea arăta astfel:

```
bpc suma\2457.pas
bpc dif\2457.pas
bpc produs\2457.pas
```

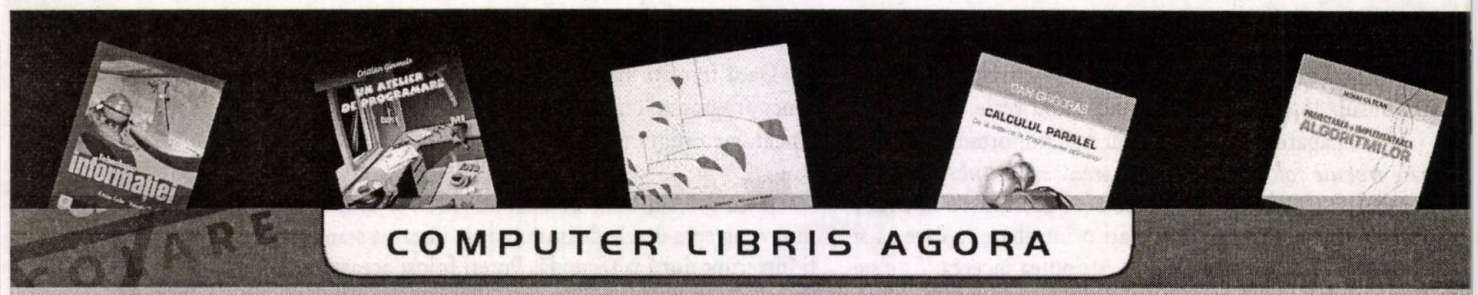
Folosiți `bpc` pentru compilarea surselor *Pascal* și `bcc` pentru compilarea surselor *C/C++*.

După crearea acestui fișier trebuie construită arhiva pe care o veți trimite. Folosiți un arhivator care creează arhive `.rar` sau `.zip`; nu creați o arhivă autoexpandabilă. Numele arhivei ar trebui să fie **2457R1.ZIP** sau **2457R1.RAR**.

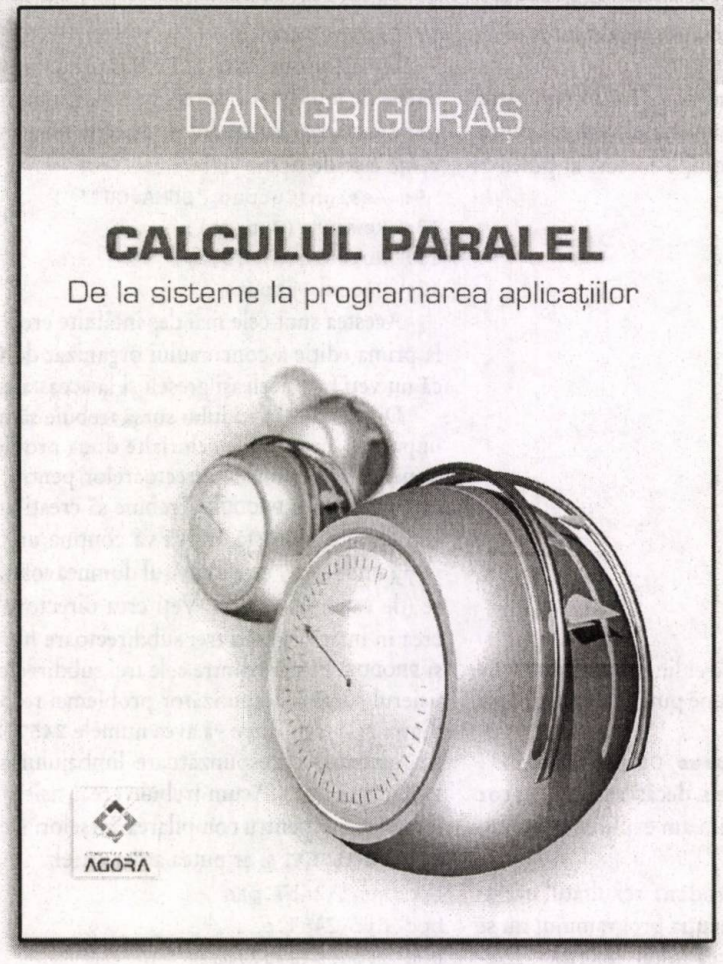
Acum puteți trimite mesajul e-mail care va conține soluțiile. Nu uitați să atașați arhiva și să scrieți în linia *Subject*: `2457 Runda 1`.

Atenție!!!

Nu folosiți numărul `2457` decât dacă este propriul dumneavoastră cod de identificare pentru concursul *Bursele Agora*. Folosiți ID-ul dumneavoastră; am folosit `2457` doar ca un exemplu.



COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

MATRICE

Denumirea subdirectorului: **MATRIX**

Denumirea fișierului de intrare: **MATRIX.IN**

Denumirea fișierului de ieșire: **MATRIX.OUT**

Descrierea problemei

Se dă o matrice cu N linii și șapte coloane. Fiecare element al acestei matrice este un număr întreg cu semn, care poate fi reprezentat pe 16 biți. Elementele pot fi numere pozitive, negative, sau pot avea valoarea 0.

Aveți la dispoziție K pătrate de dimensiuni 2 x 2. Laturile acestor pătrate sunt paralele cu laturile matricei și fiecare pătrat trebuie să acopere complet exact patru elemente ale matricei. Pătratele trebuie să se afle complet în interiorul matricei, nici una dintre liniile sau coloanele lor nu poate fi în afara ei. Astfel, colțul din stânga sus al unui pătrat nu poate fi niciodată pe a șaptea coloană sau pe ultima linie (a N-a) a matricei.

Sarcina voastră este de a acoperi elemente ale matricei folosind pătratele date astfel încât suma elementelor acoperite să fie maximă. În acest scop puteți folosi nici unul, unul, mai multe sau toate pătratele care vă stau la dispoziție. Pătratele care acoperă elementele matricei pot să se suprapună.

Suma elementelor acoperite este calculată însumând elementele care sunt acoperite de cel puțin unul dintre pătratele folosite. Chiar dacă un element al matricei este acoperit de două sau mai multe pătrate, valoarea sa este însumată ***o singură dată***.

Date de intrare

Prima linie a fișierului de intrare va conține două numere întregi, separate printr-un singur spațiu. Primul va indica numărul N al liniilor matricei, iar al doilea va indica numărul K al pătratelor pe care le aveți la dispoziție pentru a acoperi elementele matricei.

Fiecare dintre următoarele N linii va conține câte șapte numere întregi separate prin câte un spațiu. Acestea vor reprezenta cele $7 \cdot N$ elemente ale matricei. Pe prima dintre aceste linii se vor afla elementele corespunzătoare primei linii a matricei, pe cea de-a doua linie cele corespunzătoare celei de-a doua linii a matricei etc. Primul element dintr-o linie va corespunde elementului de pe prima coloană a liniei respective, al doilea va corespunde elementului de pe a doua coloană etc.

Date de ieșire

Fișierul de ieșire va conține o singură linie pe care se va afla un număr întreg care va indica suma maximă care poate fi obținută.

Restricții

- $2 \leq N \leq 100$;
- $1 \leq K \leq 100$;

Exemplu

MATRIX.IN

```
3 4
1 1 0 0 0 0 0
1 1 1 0 0 -3 0
-4 1 1 0 0 1 1
```

MATRIX.OUT

```
7
```

Explicație

Valoarea 7 se obține plasând două dintre cele patru pătrate pe care le avem la dispoziție.

Primul dintre ele are colțul din stânga-sus în prima linie și prima coloană a matricei. Cele patru elemente acoperite de el au fiecare valoarea 1, deci suma lor este 4.

Al doilea pătrat are colțul din stânga-sus în a doua linie și a doua coloană a matricei. Și de această dată cele patru elemente acoperite au fiecare valoarea 1 dar, elementul din linia a doua și coloana a doua a fost deja acoperit de primul pătrat, deci valoarea sa nu este adunată încă o dată la suma finală; așadar, suma obținută este 7.

Se observă că plasarea unui alt pătrat nu ar duce la creșterea sumei.

Timp de execuție/test: **1 secundă**

Runda 1, problema 2

DUNE

Denumirea subdirectorului: **DUNE**

Denumirea fișierului de intrare: **DUNE.IN**

Denumirea fișierului de ieșire: **DUNE.OUT**

Descrierea problemei

După moartea tatălui său, *Paul Atreides* s-a refugiat în deșert printre fremeni. Aici el a primit numele *Muad'Dib* și a început să îi organizeze pe oamenii deșertului pentru a lupta împotriva *Harkonnienilor* în vederea redobândirii de către *Casa Atreides* a controlului asupra mirodeniei, bogăția inestimabilă a planetei-deșert *Arrakis*. Doar fremenii cunoșteau adevărata identitate al lui *Muad'Dib*, dar faima sa de profet a ajuns până pe *Giedi Prim*, planeta de reședință a *Casei Harkonnen*. Baronul *Vladimir Harkonnen* și-a dat seama că ar fi bine să cunoască acțiunile celui care a devenit într-un timp foarte scurt simbolul fremenilor. În acest scop, el a reușit să infiltreze printre fremeni un spion care avea asupra sa un dispozitiv de urmărire. *Paul Muad'Dib* și-a dat seama că este urmărit și s-a gândit să plece în deșert. Totuși, spionul a reușit să îi ascundă în interiorul distraiului un aparat de urmărire rudimentar. Acest aparat este capabil de a detecta doar direcția de mers, dar nu și distanțele parcurse.

Planeta *Dune* (denumirea dată de fremeni *Arrakisului*) este descrisă printr-un tablou bidimensional de dimensiuni $m \times n$. O regiune stâncoasă este reprezentată de caracterul `'+'`, o regiune deșertică prin caracterul `'.'`, iar *sietch*-ul din care pleacă *Muad'Dib* prin caracterul `'*'`.

Se știe că pericolul principal al planetei *Dune* îl reprezintă uriașii viermi de nisip. Doar cei foarte bine antrenați pot supraviețui întâlnirii cu acești monștri care pot avea lungimi de câțiva kilometri. Totuși, trăind în deșert, fremenii au descoperit cum pot să folosească acești viermi în avantajul lor. Ținuturile stâncoase sunt controlate de către *Harkonnieni* deoarece aici nu există pericolul viermilor. Din aceste motive tânărul *Atreides* se deplasează doar prin deșert. *Baronul* a aflat acest lucru și încearcă să folosească informația în avantajul său.

Pentru aceasta îl cheamă pe mentatul *Thufir Hawat* și îi cere să determine șansele pe care le are de a-l găsi pe profetul fremenilor. În acest scop mentatul trebuie să determine numărul pozițiilor în care se poate afla *Muad'Dib*. *Hawat* știe că fremenul se poate deplasa doar pe orizontală și pe verticală și că dispozitivul de urmărire va detecta orice schimbare de direcție. După schimbarea unei direcții *Muad'Dib* se deplasează cu cel puțin o poziție în acea direcție, și modificările de direcție au loc întotdeauna cu 90°.

Date de intrare

Prima linie a fișierului de intrare conține numerele m și n, separate printr-un singur spațiu. Următoarele m linii conțin câte n caractere care descriu suprafața planetei (caracterele admise sunt `'.'`, `'+'` și `'*'`; ele au semnificația prezentată anterior). Următoarea linie va descrie informațiile furnizate de dispozitivul de urmărire, neseparate prin spații. Litera `'N'` indică o deplasare spre nord (spre prima linie a matricei), litera `'V'` o deplasare spre vest (spre prima coloană), litera `'S'` o deplasare spre sud (spre ultima linie), iar litera `'E'` o deplasare spre est (spre ultima coloană).

Date de ieșire

Fișierul de ieșire va conține un singur număr care va indica numărul pozițiilor în care se poate afla *Muad'Dib*.

Restricții

- $1 \leq m, n \leq 50$;
- numărul schimbărilor de direcție este cel mult 1000;
- *Paul* nu poate ieși în afara matricei.

Exemplu

DUNE.IN

```
4 5
.....
.+.+.
+....
.++.*
NVS
```

DUNE.OUT

```
4
```

Timp de execuție/test: **1 secundă**

WARCRAFT

enumirea subdirectorului: **WARCRAFT**

enumirea fișierului de intrare: **WARCRAFT.IN**

enumirea fișierului de ieșire: **WARCRAFT.OUT**

Descrierea problemei

După câteva sute de ani de războaie, orcii și oamenii au început tratativele de pace. În acel moment fiecare națiune controla părți din teritoriul *Azeroth*-ului.

Harta *Azeroth*-ului poate fi reprezentată cu ajutorul unei matrice de dimensiuni $m \times n$. Ținuturile controlate de către orci sunt reprezentate prin caracterul `'.'`, iar cele controlate de oameni sunt reprezentate prin caracterul `'+'`.

Totuși, în urma războaielor, teritoriile controlate de o națiune ar putea fi izolate de alte teritorii care sunt controlate de aceeași națiune. Acest lucru nu convine nației respective, deoarece acele teritorii ar putea fi foarte ușor invadate și cucerite.

Așadar, orcii și oamenii doresc crearea a două zone de control disjuncte, astfel încât să se poată deplasa între oricare două puncte ale unei zone de control fără a trece prin teritoriul fostului inamic. Pentru o deplasare dintr-un punct în altul, un călător se poate deplasa doar pe orizontală și verticală (nu și pe diagonală).

Pentru a rezolva această problemă, orcii și oamenii s-au întâlnit din nou. Un bătrân *dwarf* a propus un schimb de teritorii. Astfel oamenii vor ceda un teritoriu controlat de ei (un element al matricei care are valoarea `'+'`) în schimbul unui teritoriu controlat de orci (un element al matricei care are valoarea `'.'`).

Tinerii *elfi* și-au dat seama că un astfel de schimb ar fi echitabil și ar rezolva o mulțime dintre problemele cu care se confruntă *Azeroth*-ul. De aceea, ei au propus ca schimbul de teritorii să se realizeze în așa fel încât numărul teritoriilor care sunt schimbate să fie minim.

De asemenea, pentru a simplifica procedura și a scurta durata schimburilor, un *ogre* a propus ca un teritoriu al cărui stăpân s-a schimbat, să rămână sub controlul acestuia și nu mai poate fi recuperat în schimbul unui alt teritoriu.

Va trebui să scrieți un program care va determina numărul minim de teritorii pe care trebuie să le cedeze oamenii (evident, numărul teritoriilor cedate de orci va fi același) pentru ca în interiorul celor două zone de control formate să se poată călători între oricare două puncte fără a trece prin teritoriul inamic.

Date de intrare

Prima linie a fișierului de intrare conține numerele m și n, separate printr-un singur spațiu. Următoarele m linii conțin câte n caractere care descriu suprafața planetei (caracterele admise sunt `'.'` și `'+'` ; ele au semnificația prezentată anterior).

Date de ieșire

Fișierul de ieșire va conține un singur număr care va indica numărul minim al ținuturilor cedate de oameni (care este egal cu numărul minim al ținuturilor cedate de orci).

Restricții

- $1 \leq m, n \leq 50$;
- $\max(m, n) \geq 2$.

Exemplu

WARCRAFT.IN	WARCRAFT.OUT
4 5 ..... .+.+. +.... .++.+	2

Explicație

O posibilitate de realizare a zonelor de control respectând cerințele date este descrisă în continuare. Teritoriul controlat de oameni din linia a doua și coloana a doua este schimbat cu teritoriul controlat de orci din linia a patra și prima coloană, iar teritoriul controlat de oameni din linia a doua și coloana a patra este schimbat cu teritoriul controlat de orci din linia a patra și coloana a patra.

Timp de execuție/test: **1 secundă**

Un MODEL genetic pentru problema ORARULUI

Radu Vișinescu

Acest articol este structurat în două părți; în prima dintre ele este prezentată noțiunea de algoritm genetic, iar în cea de-a doua se propune un model de algoritm genetic în vederea rezolvării unei cunoscute probleme de programare, cea a generării orarului de funcționare a unei instituții de învățământ.

Algoritmi genetici

Introducere

Algoritmii genetici reprezintă un caz particular al algoritmilor evolutivi. În conceperea algoritmilor evolutivi s-au folosit operații specifice existente în natura la nivelul secvențelor de ADN ce formează cromozomii organismelor vii. Legat de aceste operații există și unele experimente care iau în considerare un nou tip de calculator pe bază de *ADN* [3].

Algoritmii genetici au apărut din dorința de a modela pe calculator diverse situații evolutive și pentru testarea în cadrul acestora a diferitelor ipoteze. De asemenea, o puternică motivație a fost rezolvarea pe această cale a problemelor NP (nedeterminist polinomiale) [2].

Prin modelarea situației din natură putem vorbi despre ***cromozom*** ca fiind un vector de valori de lungime finită. Un element al vectorului cromozom se va numi ***genă***, iar o genă poate lua la rândul său o mulțime finită de valori. În multe cazuri, similar unui bit, gena poate lua valori din mulțimea {0, 1}.

Operația de recombinare a doi cromozomi de aceeași lungime poartă în literatură denumirea de ***crossover***. Are ca rezultat alți doi cromozomi obținuți din secvențe analoage ale cromozomilor inițiali (părinți).

Prezentăm un caz particular al acestei operații în care cromozomii se "taie în două locuri" ($k = 2$).

Fie $k = 2$ și $1 \le t_1 < t_2 \le n$. Din cromozomii

$c_i = x_1x_2\cdots x_{t1}x_{t1+1}\cdots x_{t2}x_{t2+1}\cdots x_N$

și

$c_j = y_1y_2\cdots y_{t1}y_{t1+1}\cdots y_{t2}y_{t2+1}\cdots y_N$

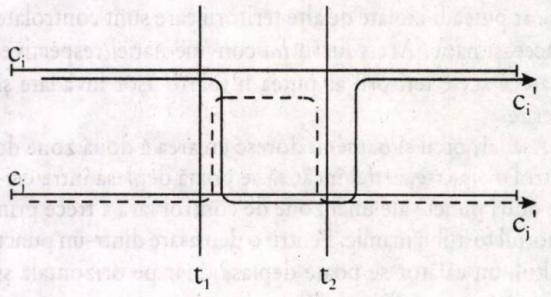
se vor obține cromozomii:

$c_i' = x_1x_2\cdots x_{t1}y_{t1+1}\cdots y_{t2}y_{t2+1}\cdots y_N$

și

$c_j' = y_1y_2\cdots y_{t1}x_{t1+1}\cdots x_{t2}x_{t2+1}\cdots x_N$.

Operația este prezentată grafic în figura următoare:



Structura unui algoritm genetic

Prezentăm cazul de bază al aplicării unui algoritm genetic.

Problema constă în aflarea valorii maxime a unei funcții de două variabile și a punctului în care se obține această valoare. Pe caz general avem: $f:(a, b) \to R$; se poate verifica faptul că ne putem restrânge la cazul în care intervalul (a, b) este intervalul $(0, 1)$.

Considerăm o valoare întreagă pozitivă n, de exemplu $n = 20$. Considerăm cele 2^n șiruri de lungime n cu elemente 0 și 1. Un astfel de șir poate fi considerat ca fiind partea fracționară în baza doi a unui număr din intervalul (0,1). (Pentru $n = 2$ avem: 00 este 0, 01 este 1/4, 10 este 1/2, iar 11 este 3/4). Putem considera că cele 2^n șiruri reprezintă tot atâtea puncte din intervalul (0, 1).

Un astfel de șir îl vom numi ***cromozom***.

O ***genă*** va fi o valoare (0 sau 1) de pe o anumită poziție.

Putem descrie acum structura generală a algoritmului genetic în pseudocod:

```
creare_populaþie_iniþialã(n,nr)          // pas 1
g ← 0
repetã
  g ← g+1
  selectare_populatie_intermediarã()     // pas 2
  mutaþii_încrucisate(pc);               // pas 3
  mutatii_simple(ps)                     // pas 4
pânã când g=gmax
afisare_rezultat()                       // pas 5
```

Pasul 1 - crearea populației inițiale

Considerăm o valoare *nr*, spre exemplu *nr* = 100, și generăm aleator un număr de *nr* cromozomi (șiruri) de lungime *n*. Numim această mulțime de șiruri populația inițială de cromozomi; *n* și *nr* sunt doi parametri ai algoritmului.

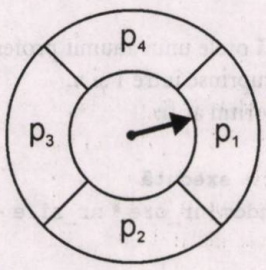
Vom analiza în cele ce urmează pașii din interiorul buclei. La fiecare iterație se trece de la generația curentă la generația următoare; *gmax* este un al treilea parametru reprezentând numărul maxim de generații.

La început generația curentă este populația inițială de cromozomi.

Pasul 2 - selectarea populației intermediare

Se evaluează populația curentă. În cazul de față aceasta constă în calcularea valorilor funcției *f* în cele *nr* puncte. Fie $v_i := f(c_i)$ pentru $i = 1, 2, ..., nr$. Un cromozom se va numi ***valoros*** dacă valoarea funcției în respectivul punct este mare. Fie $S := v1 + v_2 + ... + v_{nr}$. Considerăm probabilitățile $p_i := v_i / S$;

Aplicăm un mecanism de tip ruletă rusească prin care selectăm cu probabilitate mai mare cromozomii valoroși. Cu alte cuvinte, la fiecare pas vom genera un număr aleator între 0 și 1 și calculăm în ce zonă de probabilitate ne aflăm. Această operație este ilustrată grafic în figura următoare:



Ruleta va avea sloturi proporționale cu probabilitățile p_i, $i = 1, 2, ..., nr$ și o vom "întoarce" de *nr* ori .

Pasul 3 - aplicarea mutației încrucișate

Unui procent din populația intermediară i se vor aplica mutații încrucișate. Pentru a selecta acești cromozomi se stabilește un al patrulea parametru și anume *pc* (probabilitatea de mutație încrucișată) - are valoarea de aproximativ 0,1. Pentru fiecare cromozom al populației intermediare generăm un număr aleator între 0 și 1. Dacă acest număr este mai mic decât *pc*, cromozomul va fi selectat pentru mutații încrucișate.

Mutația propriu-zisă este de tip *crossover* și se realizează așa cum se va descrie în continuare.

Fie cromozomii selectați:

$c_i = x_1x_2...x_N$

$c_j = y_1y_2...y_N$

Generăm aleator un număr *t* din mulțimea 1, 2, ..., *n*.

Cei doi cromozomi se vor înlocui cu următorii doi:

$c_i' = x_1x_2...x_ty_{t1+1}...y_N$

$c_j' = y_1y_2...y_tx_{t1+1}...x_N$

Pasul 4 - aplicarea mutației simple

Unui procent (mai mic) din populația intermediară, procent ce depinde de un parametru *ps* (probabilitate de mutație simplă), i se vor aplica mutații simple. Procedeul are loc așa cum se descrie în continuare.

Se va genera aleator un număr *t* din mulțimea 1, 2, ..., *n*. Conținutul genei de pe poziția *t* se va modifica.

Observație

O caracteristică a algoritmilor genetici este că lucrează cu întrega populație; contează prea puțin cum evoluează indivizii, important este ca populația (luată ca un întreg) să evolueze. Nimeni nu garantează că o nouă generație va fi neapărat mai valoroasă, această evoluție are lor ca urmare a unor legi statistice.

În final se va ajunge la o populație care nu va mai evolua. Ultimul pas (al cincilea) este executat o singură dată.

Pasul 5 - selectarea rezultatului

După *gmax* generații, maximul funcției pe intervalul (0, 1) va fi aproximat de cel mai valoros cromozom din ultima generație. Acesta va fi afișat ca rezultat.

Analiza complexității

Crearea populației inițiale prin randomizare poate fi efectuată într-un timp de ordinul $O(n \cdot nr)$.

Presupunem că evaluarea funcției într-un punct necesită un timp constant. Luând în considerare și durata de calcul al punctului din intervalul (0, 1) asociat cromozomului, evaluarea populației curente are loc într-un timp de ordinul $O(n \cdot nr)$.

"Întoarcerea ruletei" se efectuează într-un timp de ordinul $O(nr)$.

Mutațiile de orice tip sunt realizate în timp constant, deci și pentru mutații avem un timp de ordinul $O(nr)$.

Rezultă că bucla repetitivă este executată într-un timp de ordinul $O(n \cdot nr \cdot gmax)$.

În concluzie, algoritmul este polinomial în parametrii *n*, *nr* și *gmax*. Observăm că durata de execuție este direct proporțională cu parametrul *gmax* (numărul maxim de generații).

Cazuri nedorite

Este posibil ca algoritmul să se "blocheze" în maxime locale. Pentru evitarea acestei probleme se recomandă respectarea următoarelor cerințe:

1. Alegerea de funcții de test semnificative;
2. Testarea algoritmilor utilizând și un alt generator (propriu) de numere aleatoare pe lângă apeluri ale funcției `random()`.
3. Alegerea parametrilor *n*, *nr*, *gmax*, *pc*, *ps* astfel încât să se obțină o bună convergență.

Indicație

Problema minimizării unei funcții *f* se reduce la maximizarea funcției *-f*.

Problema orarului

Descrierea problemei

Enunțul general al problemei este următorul [1]:

Se dă matricea încadrărilor, precizând pentru fiecare profesor numărul de ore la fiecare clasă. Cerința principală este aceea de a se obține un orar, adică o repartizare a profesorilor la clase, pentru fiecare dintre zilele săptămânii și, în cadrul fiecărei zile, pentru fiecare oră, astfel încât într-o aceeași oră (dintr-o anumită zi a săptămânii) fiecare profesor să intre la cel mult una din clasele asociate lui prin matricea de încadrare și la fiecare clasă să intre cel mult câte unul din profesorii asociați clasei.

În continuare vom considera un caz simplificat al acestei probleme, caz ce acoperă totuși o mulțime de situații concrete.

Vom considera următoarele date de intrare:

- *nr_cl*: numărul de clase din școală;
- *nr_ore*: numărul maxim de ore pe zi ale unui elev;
- *nr_zile*: numărul de zile în care sunt ore;
- *nr_p*: numărul de profesori;
- încadrarea profesorilor dată printr-o matrice cu *nr_p* linii și *nr_cl* coloane numită matricea încadrărilor notată în continuare cu *A*. Profesorii vor fi codificați cu numere $i = 1, ..., nr\_p$ iar clasele cu numere $j = 1, ..., nr\_cl$;
- elevii din clasele 9-10 învață după-amiaza, iar cei din clasele 11-12 dimineața.

Exemplu

- *nr_cl* = 40; 40 de clase - acestea ar putea fi clasele 9A, ..., 9H, 10A, ..., 10H, 11A, ..., 11H, 12A, ..., 12H;
- *nr_ore* = 6 (șase ore pe zi);
- *nr_zile* = 5 (cinci zile pe săptămână);
- *nr_p* = 80 (80 de profesori - se păstrează și o listă cu numele lor: Aldea, Andrei, ..., Voicu);
- o linie a matricei încadrărilor ar putea indica, de exemplu, faptul că profesorul Aldea are următoarele ore: 5 ore la 10B, 4 ore la 11A, 3 ore la 12 C, 3 ore la 12D, 3 ore la 12E.

Orarul de funcționare se poate modela printr-o matrice cu nr_p linii și $2 \cdot nr\_ore \cdot nr\_zile$. Indicele liniei reprezintă codul profesorului, iar indicele coloanei un anumit moment de timp (zi și oră). Un element al acestei matrice va fi codul unei clase. O linie a matricei va indica programul pe zile al unui profesor, iar o coloană va indica profesorii ce au ore la momentul respectiv.

Condițiile pe care trebuie să la îndeplinească un orar de funcționare sunt următoarele:

- trebuie să fie conform cu încadrarea (linia de orar trebuie să exprime exact și încadrarea profesorului);
- la un anumit moment de timp (zi și oră), la o anumită clasă trebuie să predea cel mult un profesor;
- un profesor nu poate preda simultan la mai multe clase;
- elevii nu vor avea ferestre (ore libere între două ore la care li se predă).

Vom numi un astfel de orar ca fiind un ***orar valid.***

Pe lângă cele patru condiții absolut necesare ar fi de dorit ca profesorii să aibă un număr cât mai mic de ferestre, iar elevii să aibă cel mult două ore la aceeași disciplină în aceeași zi.

Un orar valid cu număr mic de ferestre pentru profesori va fi numit un ***orar bun.***

O posibilă rezolvare

Pentru început va trebui să găsim un orar valid de pornire. Există mai multe posibilități de determinare a unui asemenea orar.

O primă variantă poate fi găsită în [1]; trebuie menționat faptul că s-ar putea ca prin colorarea orelor și ulterior a *S*-orelor, să nu se obțină de la început o repartizare corectă (un orar valid).

De asemenea, se poate observa că obținerea *S*-orelor este realizată cu ajutorul unui algoritm care folosește metoda *backtracking*, deci consumă mult timp.

Putem lua în considerare o altă variantă de generare a unui asemenea orar de pornire.

Concret, folosind funcția `Random`, se vor așeza în matricea orarului, linie cu linie, pozițiile în care respectivul profesor ține orele prevăzute în încadrare.

Pentru fiecare linie se va utiliza un algoritm de generare a unui aranjament aleator de ore conform cu matricea încadrărilor.

Presupunem că orele unui anumit profesor sunt codificate cu numere cuprinse între 1 și *n*.

Un posibil algoritm ar fi:

```
pentru pas ← 1,n executã
  aleator ← random(nr_ore * nr_zile - pas + 1) + 1
  poz ← 0
  nr ← 0
  cât timp nr < aleator executã
    poz ← poz+1
    dacã a[poz]=0 atunci
      nr ← nr+1
    sfârsit dacã
  sfârsit cât timp
  a[poz] ← pas
sfârsit pentru
```

Inițial se vor "masca" pozițiile de dimineață pentru clasele 9-10 și pozițiile de după-amiază pentru clasele 11-12. Apoi, în raport cu numerele aleatoare generate, se va uti-

liza o mască de dimensiunea orarului. Pe fiecare linie pozițiile deja ocupate cu ore la o anumită clasă vor fi "mascate" pas cu pas.

Acestă generare este cu atât mai abordabilă cu cât vor exista clase care învață un număr de ore pe săptămână mai mic decât numărul maxim de ore prevăzut.

După generarea orarului de pornire, acesta trebuie optimizat în raport cu numărul de ferestre.

Un orar valid poate fi considerat ca fiind o matrice B cu nr_p linii și $2 \cdot nr\_ore \cdot nr\_zile$. Pentru exemplul considerat vor fi 80 de linii și 60 de coloane. 30 dintre coloane vor reprezenta ore de dimineață (cele care au un indice care împărțit la 12 dă un rest cuprins între 1 și 6), iar celelalte 30 ore de după-amiază. Un element al acestei matrice va fi codul unei clase (de la 1 la nr_cl) sau valoarea 0 dacă pe linia profesorului p, la timpul t nu este planificată desfășurarea unei ore.

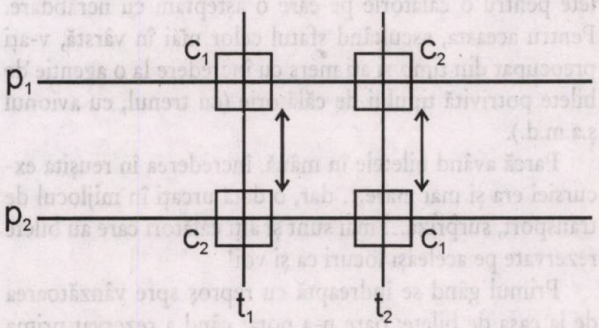
O linie a matricei B va fi considerată cromozom. Pentru exemplul ales, lungimea cromozomului este 60.

O genă va putea lua valori cuprinse între 0 și nr_cl. Vor putea fi 41 de valori distincte pentru gene.

Dimensiunea populației este egală cu nr_p. În cazul acestui exemplu, valoarea este 80.

Ca mutații încrucișate putem considera următorul tip de *crossover*:

- fie profesorii p_1 și p_2 care la timpii t_1 și t_2 au ore la clasele c_1 și c_2; cu alte cuvinte vom avea $B(p_1, t_1) = B(p_2, t_2) = c_1$ și $B(p_1, t_2) = B(p_2, t_1) = c_2$; cromozomii corespunzători liniilor p_1 și p_2 se vor încrucișa interschimbând genele de pe pozițiile t_1 si t_2; operația poate fi reprezentată grafic astfel:



Observăm că prin astfel de mutații se păstrează validitatea orarelor.

Este greu de găsit un tip de mutație simplă care să conserve proprietatea unui orar de a fi valid .

Funcția de minimizat este funcția care calculează numărul de ferestre al unui profesor.

Această funcție se poate eventual pondera, în sensul ca o fereastră de două ore să fie echivalentă, de exemplu, cu trei ferestre de câte o oră.

Construirea unui orar bun va însemna optimizarea prin algoritmul genetic a unui orar valid de pornire. Optimizarea va avea ca scop reducerea numărului de ferestre ale profesorilor.

Observații

- Algoritmul descris poate fi implementat astfel încât să se execute într-un timp de ordinul $O(nr\_p \cdot nr\_ore \cdot nr\_zile \cdot gmax)$.
- Conform celor prezentate anterior, putem varia valorile parametrilor pc și $gmax$ în vederea obținerii unei bune convergențe (mai puține generații construite până la obținerea unui orar bun).
- Este esențială testarea algoritmului pe date de dimensiune reală, deoarece convergența algoritmilor genetici depinde, într-o măsură destul de mare, de valorile parametrilor n și nr.

În cele ce urmează vom prezenta câteva modificări care ar putea fi aduse algoritmului prezentat. Fie M_0, M_1, M_2,... șirul de orare generate de algoritm. M_g va reprezenta populația din generația g. În scopul îmbunătățirii performanțelor pot fi luate în considerare și următoarele modificări la descrierea generală a algoritmilor genetici.

O posibilitate de optimizare implică introducerea a încă doi parametri întregi pozitivi p și $delta$.

Fie g generația curentă. În cadrul buclei repetitive se vor executa următoarele operații:

- salvarea populației corespunzătoare generației curente g (M_g);
- generarea a $delta$ noi populații conform algoritmului prezentat; noile orare vor fi identificate prin M_{g+1}, M_{g+2}, ..., $M_{g+delta}$;
- păstrarea celei mai valoaroase generații, care va deveni generația $M_{g+delta}$;
- creșterea numărului de generații: $g \leftarrow g + delta$.

O a doua posibilitate de optimizare implică introducerea parametrului întreg $delta$ și a unui parametru S care este un șir de numere naturale.

Fie S un șir descrecător de numere naturale care conține $gmax$ elemente. În cadrul buclei repetitive se vor executa următoarele operații:

- salvarea populației corespunzătoare generației curente g (M_g);
- generarea a $delta$ noi populații conform algoritmului prezentat; noile orare vor fi identificate prin M_{g+1}, M_{g+2}, ..., $M_{g+delta}$;
- repetarea pasului anterior până când numărul total de ferestre scade sub valoarea S_g; populația corespunzătoare devine generația $M_{g+delta}$;
- creșterea numărului de generații: $g \leftarrow g + delta$.

Bibliografie:

[1]. *Vlad Bazon*, *Un model pentru problema orarului*, GInfo 9/8 (decembrie 1999);

[2]. *Ricardo Poli*, *Genetic algorithms*, http://www.cs.bham.ac.uk/~rmp/slide_book/node2.html;

[3]. *George Păun*, *Calculatoare pe bază de ADN*, GInfo 9/3 (martie 1999).

Dl. prof. Radu Vișinescu este cadru didactic la Liceul "Ion Luca Caragiale" din Ploiești. Poate fi contactat prin e-mail la raduv@lphilc.sfos.ro.

Introducere în problematica sistemelor NESECVENȚIALE

Anca Vasilescu, Oana Georgescu

În acest serial ne propunem să parcurgem împreună câteva elemente definitorii ale conceptelor de concurență, paralelism și distribuire, concepte foarte moderne în programare și în teoria sistemelor. Aceste elemente vor fi alese cu grijă, astfel încât să vă trezească interesul și curiozitatea pentru această lume fascinantă.

Pentru început, pentru că sunteți proaspăt întorși din vacanță (și tocmai ați renunțat la modelul distribuit al clasei voastre pentru unul puternic cuplat, în incinta școlii) să amânăm abordarea standard: definiție, definiție, propoziție, teoremă, demonstrație... și să alegem câteva probleme clasice care confirmă că este necesar ca informaticianul să aibă la îndemână pentru modelarea lumii înconjurătoare și alte mecanisme decât cele secvențiale. (Un sistem este secvențial dacă activitatea lui se poate împărți în etape și dacă aceste etape se succed în timp într-o ordine unică, predeterminată algoritmic.)

Jocul vieții

S-ar putea ca această problemă clasică de concurență să vă fie deja cunoscută din [1, pag. 113]. Pentru scopul acestui serial preluăm formularea problemei din [2]:

Un tabel dreptunghiular, notat A, cu m linii și cu n coloane conține celule care pot fi vii sau moarte. Fiind dată o configurație curentă a tabelului, se cere să se determine configurația generației următoare, știind că celulele moarte își păstrează starea, iar o celulă vie moare dacă și numai dacă numărul celulelor vii, vecine ei este mai mic decât 2 sau mai mare decât 4.

O celulă are cel mult 8 vecini: pe orizontală, pe verticală și pe diagonale.

Preferăm să nu comentăm acum această problemă, dar vă invităm să consultați problema 24 din [1, pag. 113] și să încercați să o rezolvați cu mijloacele și structurile capitolului respectiv.

Apoi, după ce ați citit acest serial, vă propunem să o rezolvați ținând cont de elementele specifice pe care le-ați învățat.

Problema rezervării biletelor

Fiecare dintre noi am fost puși în situația de a cumpăra bilete pentru o călătorie pe care o așteptam cu nerăbdare. Pentru aceasta, ascultând sfatul celor mai în vârstă, v-ați preocupat din timp și ați mers cu încredere la o agenție de bilete potrivită tipului de călătorie (cu trenul, cu avionul ș.a.m.d.).

Parcă având biletele în mână, încrederea în reușita excursiei era și mai mare... dar, o dată urcați în mijlocul de transport, surpriză... mai sunt și alți călători care au bilete rezervate pe aceleași locuri ca și voi!

Primul gând se îndreaptă cu reproș spre vânzătoarea de la casa de bilete: oare n-a notat când a rezervat prima dată locurile? Dar dacă mai multe vânzătoare aveau posibilitatea să rezerve aceleași locuri și n-au apucat să se anunțe între ele cine a vândut prima biletele respective? Acum, după plecarea trenului, ar ști să ne spună care locuri au rămas libere și câte bilete s-au vândut?

Această situație, din păcate întâlnită destul de des, este modelul pentru problema rezervării biletelor. O formulare completă ar fi următoarea [2]:

Fiecare terminal al unei rețele de calculatoare este plasat într-un punct de vânzare a biletelor pentru un anumit scop (spectacol, călătorie etc.). Se cere să se găsească o modalitate de a evita vânzarea mai multor bilete pentru același loc.

Acesta este un exemplu de ***paralelism*** deoarece, în același timp (deci în paralel) mai multe case de bilete (terminalele rețelei) vând bilete pentru aceleași locuri.

În plus, apare și ***concurența***, deoarece mai multe ***procese*** (*vânzătoarele de bilete*) încearcă să folosească în același timp un element comun, o ***resursă*** (inițial) comună și anume locul din tren sau din sala de spectacol.

În particular, aici resursa *loc* nu poate fi ***partajată*** (folosită în comun, în același timp, de mai multe procese) deoarece aceasta ar conduce la situația greșită în care mai multe vânzătoare au rezervat același loc pentru călători diferiți.

O astfel de entitate care poate fi folosită exclusiv de un proces la un moment dat este o ***resursă critică***. De exemplu, imprimanta folosită în rețea de mai mulți utilizatori este o resursă critică.

Ce s-ar întâmpla dacă, de exemplu, atunci când imprimanta primește ***simultan*** mai multe fișiere de listat, ea ar lista un singur document în care să ar prelua câteva linii dintr-unul din documentele de intrare, apoi câteva linii din altul, apoi iar câteva din primul, ș.a.m.d?...

Și în acest caz se impune ca, o dată ce imprimanta a început listarea unui document, ea să nu mai preia altă comandă de listare până când nu-și încheie execuția comenzii anterioare.

Un alt exemplu de resursă critică frecvent utilizată de dumneavostră este compilatorul unui mediu de programare folosit în rețea, în același timp, de mai mulți programatori.

Revenind la problema rezervării biletelor, rezolvarea concurenței și implicit a problemei, presupune folosirea unor mecanisme (unelte hard și soft) specializate care să asigure accesul exclusiv al unui proces (*vânzătoare*) la o resursă critică (*loc*) la un moment dat.

Cu aceste mecanisme ne vom întâlni în episoadele următoare.

Problema cititor-scriitor

Problema cititor - scriitor se referă la organizarea unui set de date (o *carte*) când acesta este accesat de procese de tip cititor și scriitor.

Astfel, cartea devine o resursă comună la care au acces diferite procese. Setul de date se numește de obicei *carte* pentru a păstra contextul sugerat de denumirile *cititor* și *scriitor*.

În acest context, un *cititor* este orice proces care poate consulta o resursă simultan cu alte procese *cititor*, dar fără a putea opera modificări asupra resursei respective.

În același timp, un *scriitor* este orice proces care trebuie să aibă acces exclusiv la resursa comună, având și drepturi de modificare asupra resursei respective.

Cu alte cuvinte, este permis ca mai mulți *cititori* să consulte cartea simultan, dar se impune ca atunci când un *scriitor* modifică conținutul cărții, nici un alt proces (indiferent dacă este *cititor* sau *scriitor*) nu trebuie să poată avea acces la carte.

Folosind elementele introduse intuitiv la problema rezervării biletelor, putem spune că în această problemă *cartea* este o resursă critică și că accesul la resursa critică de tip *carte* este permis exclusiv unui *scriitor* sau, în paralel, mai multor *cititori* simultan.

Procesele *cititor* și *scriitor* au de executat activități fundamental diferite, care trebuie ordonate corect pentru execuție. Pentru aceasta, proceselor *cititor* și *scriitor* li se pot asocia anumite reguli de prioritate pentru execuție.

Situațiile de concurență apar când *cartea* este solicitată simultan atât de procese *cititor* cât și de procese *scriitor*. In aceste cazuri programatorul are la dispoziție mai multe opțiuni pentru ordonarea intrării în execuție a proceselor:

- când un *scriitor* solicită accesul la carte, se întrerupe (sau nu) activitatea tuturor *cititorilor* activi astfel încât să poată fi actualizat cu prioritate conținutul *cărții*?
- programatorul întrerupe (sau nu) un *scriitor* care tocmai scrie, în favoarea unui *scriitor* care solicită accesul la *carte*?
- dacă sunt mai multe procese (atât *cititor*, cât și *scriitor*) care așteaptă să primească accesul la *carte*, care este ordinea în care intră acestea în execuție în momentul eliberării *cărții*?

Dacă veți dori să programați o astfel de problemă, una dintre primele acțiuni va fi să vă hotărâți cum veți trata aceste situații.

Dacă, extinzând modelul la alte exemple, considerăm că resursa comună este un fișier de pe disc, atunci ordinea execuției proceselor trebuie să respecte regula: informația citită trebuie să fie cea mai actuală. Pentru aceasta putem presupune că, de îndată ce un *scriitor* cere să acceseze fișierul (resursa comună), acesta va primi controlul imediat ce toți cititorii curenți (procesele cititor active) și-au finalizat acțiunile, fără ca vreun cititor să mai primească accesul între timp.

Un alt model pe care probabil că îl utilizați frecvent este accesul la un site *Web*. Aici fiecare pagină a site-lui este o resursă critică de tip *carte*, pe care o pot accesa oricâte browser-e simultan (procese client, de tip *cititor*), dar atunci când webmaster-ul (*scriitorul*) modifică conținutul unei pagini, nimeni nu poate citi informația acelei pagini.

Problema producător-consumator

Această problemă modelează situația generală în care avem o succesiune de locații (de exemplu un buffer de memorie sau o bandă rulantă) și două tipuri de procese care au acces la conținutul acelor locații: procese de tip *producător* și respectiv *consumator*.

Un proces *producător* are numai dreptul de a depune un element într-o locație a bufferului, în timp ce un proces *consumator* are numai dreptul de a prelua conținutul unei locații.

Față de problema anterioară în care numai scriitorul avea acces exclusiv la resursa comună, aici și *producătorul* și *consumatorul* trebuie să acționeze individual, la momente de timp diferite, asupra conținutului bufferului. Cu alte

cuvinte, modelarea corectă a problemei trebuie să asigure că atât *producătorul* cât și *consumatorul* au acces exclusiv la resursa comună (buffer) pe durata acțiunii lor specifice (producătorul de a pune obiecte pe bandă și consumatorul de a lua obiecte de pe bandă).

Problema trebuie tratată diferit în funcție de proprietățile particulare pe care le are bufferul în ceea ce privește dimensiunea lui, politica de ocupare cu date, protocolul de acces la date ș.a.m.d.

De exemplu, se poate presupune că bufferul este de dimensiune fixă, adică mărginit sau, mai mult, că "următoarea" locație de accesat este determinată ciclic. De obicei se consideră cazul în care locațiile sunt numerotate și un proces are acces la o anumită locație unic determinată în funcție de operația pe care procesul respectiv o solicită și, eventual, în funcție de operația anterioară care s-a executat asupra bufferului.

De asemenea, putem pune în evidență cazuri particulare de probleme în funcție de numărul proceselor care au acces la bufferul de date. Astfel, putem avea situația mai simplă cu un *producător* și un *consumator* sau, o situație mai generală, cu mai mulți *producători* și mai mulți *consumatori*.

O altă direcție de generalizare rezultă dacă se consideră că un *producător* poate depune pe bandă la un acces mai multe produse și un *consumator*, de asemenea, poate lua de pe bandă la un acces mai multe produse.

În acest model concurența apare deoarece atât *producătorul* cât și *consumatorul* accesează același buffer de date. În plus, apare și necesitatea sincronizării acțiunilor proceselor *producător* și *consumator*.

Aceasta presupune o ordonare în timp a execuției proceselor respective astfel încât să fie îndeplinite condițiile particulare impuse pentru fiecare problemă în parte. În același timp, sincronizarea se impune și pentru a trata cu atenție două situații speciale și anume:

- cazul în care un *producător* încearcă să depună un obiect, dar bufferul este plin, deci nu are unde să îl depună;
- cazul în care un *consumator* încearcă să ia un obiect, dar bufferul este gol.

Importanța acestor situații "limită" reiese din exemplul următor. Să considerăm că la un moment dat un *producător* primește accesul exclusiv la buffer și încearcă să depună obiectul său, dar nu găsește nici o locație liberă în buffer. Mai mult, aplicația este scrisă astfel încât orice acțiune asupra bufferului este suspendată până când *producătorul* transmite un semnal prin care înștiințează sistemul că și-a încheiat activitatea. Dar, bufferul fiind plin, *producătorul* nu-și poate depune obiectul și, implicit, nu ajunge să-și încheie activitatea.

În acest moment apare o configurație nedorită a aplicației și anume situația de interblocare: procesele dețin resursele de așa o manieră încât nici unul nu-și mai poate continua activitatea. Într-adevăr, *producătorul* nu-și poate continua activitatea pentru că bufferul este plin, în timp ce nici un alt proces nu primește acces la buffer până când *producătorul* nu cedează dreptul de folosire a bufferului.

În acest exemplu simplu interblocarea poate fi ușor evitată dacă programatorul dă controlul *producătorului* numai dacă bufferul nu este plin.

Acest model general poate fi "îmbrăcat" în haina interesantă a ***problemei bărbierului somnoros*** [2]:

Prăvălia unui bărbier este formată din două camere: cea de la stradă, folosită ca sală de așteptare și cea din spate, în care se găsește scaunul pe care se așează clienții pentru a fi bărbieriți. Dacă nu are clienți, bărbierul somnoros se culcă. Să se simuleze activitățile care se desfășoară în prăvălia bărbierului.

Aceasta este o reformulare a problemei *producător-consumator* în care locul bufferului de obiecte este luat de scaunul bărbierului, iar consumatorul este bărbierul, în sensul că își bărbierește (consumă) clienții.

Problema grădinii ornamentale

Deși ascunde un model teoretic foarte serios, această problemă poate fi prezentată într-o formă foarte atractivă [2]:

Intrarea în grădinile ornamentale ale unui oraș oriental se face prin N porți. Se pune problema să se țină evidența numărului persoanelor care au intrat în grădină.

Fiecare poartă de intrare în grădină este o resursă care trebuie accesată exclusiv de un proces (de o persoană care dorește să intre în grădină).

Dacă, la un moment, dat pe una dintre porți intră o persoană, atunci în acel moment pe nici o altă poartă nu mai intră vreo persoană în grădină.

Această problemă practică este cea mai apropiată de formularea teoretică, fundamentală, cunoscută ca ***problema excluderii reciproce***, pe care o vom lua în discuție în numerele viitoare.

Problema alocării resurselor

Sunt disponibile **max** *unități dintr-o resursă oarecare. Există* **nc** *clienți care doresc să aibă de mai multe ori acces la resursa respectivă, un acces constând în a lua (a încerca să ia) un număr de unități din resursă, de a folosi aceste unități și apoi de a le returna. Se cere să se simuleze aceste activități.*

Această problemă este una teoretică, foarte importantă, a cărei rezolvare face obiectul unor capitole speciale de matematică aplicată, cum ar fi *Cercetările Operaționale* [3].

Modelul problemei este foarte important, el generalizând o problemă de concurență foarte veche și anume ***problema filosofilor chinezi***, cunoscută și sub denumirea de ***problema celor cinci filosofi*** sau problema ***filosofilor la masă***.

Trecând peste ineditul numelui, să vedem la ce se referă această problemă [2]:

La o masă rotundă stau cinci filosofi chinezi. Principala lor activitate este aceea de a gândi, dar, evident, din când în când trebuie să și mănânce, folosind pentru aceasta câte două bețișoare. Știind că între oricare doi filosofi se află un bețișor și că un filosof poate mânca doar dacă a ridicat de pe masă atât bețișorul din stânga sa, cât și pe cel din dreapta sa, se cere să se simuleze activitățile filosofilor așezați la masă.

Deja din formularea problemei se poate observa că o restricție importantă care se impune este ca la nici un moment de timp să nu avem doi filosofi vecini care să mănânce.

Problema este un caz particular al problemei alocării resurselor în care avem un număr finit de procese (filosofii mâncând) care "împart" un număr finit de resurse (bețișoare), fiecare fiind folosită numai de un proces la un moment dat. O soluție secvențială imediată ar fi următoarea:

```
IaBețișor:
  IaBețișorStâng
  IaBețișorDrept
    Mănâncă
  AșeazăBețișorStâng
  AșeazăBețișorDrept
EndIaBețișor.
```

Pentru această soluție poate foarte ușor să apară ***interblocarea***: dacă toți filosofii doresc să mănânce simultan atunci fiecare ridică bețișorul din stânga sa și așteaptă ca cel din dreapta să elibereze bețișorul pentru ca el să-și poată continua activitatea (adică să aibă în dreapta bețișor pe care să-l ia). Dar cel din dreapta așteaptă același lucru de la cel din dreapta sa și așa mai departe. Astfel, toți filosofii sunt blocați în așteptare și nu se pot continua activitățile.

De aici se vede că această soluție nu este corectă. Ca și alte probleme de concurență, și aceasta trebuie abordată folosind mecanisme specifice intercomunicării între procese.

Problema emițător - receptor

Prezentarea acestei probleme este o ocazie bună pentru noi să aducem în discuție o serie de concepte foarte moderne din sfera rețelelor și aplicațiilor destinate comunicării. În plus, este timpul să spunem câteva cuvinte și despre caracterul distribuit al aplicațiilor nesecvențiale. Modelul general al problemei emițător-receptor este următorul [2]:

Un emițător emite succesiv mesaje, fiecare dintre ele trebuind să fie recepționat de toți receptorii, înainte ca emițătorul să transmită mesajul următor. Se cere să se simuleze această succesiune de activități.

Un caz particular pentru această problemă ar fi cel în care emițătorul este "grăbit" și emite mesajul numai acelor receptori care la momentul emisiei sunt gata să îl primească; totuși, și în acest caz se cere ca cel puțin un receptor să primească mesajul.

Acest caz particular servește pentru a deosebi mai multe tipuri de comunicare (transmitere de date prin mesaje) în funcție de numărul și proprietățile destinatarilor (receptorilor). Astfel, așa cum vom vedea în episoadele următoare, putem avea comunicare *broadcast*, *multicast* sau *unicast*.

În general, problema emițător-receptor este importantă în sistemele în care expeditorul (emițătorul) și destinatarii (receptorii) sunt situați la distanță. În acest caz avem un sistem distribuit în care comunicarea între nodurile din sistem se face prin transmitere de mesaje.

În general, un sistem distribuit este o colecție de procese (emițător, receptor, client, server etc.) împreună cu un subsistem de comunicație între procesele respective. Toate relațiile dintre procese sunt decise pe bază de ***negocieri*** realizate prin schimburi de mesaje.

Am amintit mai sus tipurile de procese server și client. Să spunem câteva cuvinte introductive despre acestea.

Un *proces server* este unul care se oferă să facă servicii altor procese din sistem. Cu alte cuvinte, un server oferă servicii sau produce resurse clienților săi.

În același timp, un *client* este un proces care solicită servicii de la server și le consumă.

Acest model funcționează în toate magazinele: proprietarul magazinului este serverul care oferă produse (resurse), în timp ce cumpărătorul este clientul care consumă (cumpără) produsele oferite de proprietar.

Orice aplicație în care solicitantul acțiunii este un proces și executantul acțiunii respective este un alt proces este o ***aplicație client-server*** [4].

Dacă procesele client și server nu sunt situate pe aceeași mașină (pe același calculator) atunci modelul client-server deservește sistemul distribuit și aplicația devine una distribuită. Este important de subliniat că acest model presupune implicit că, clientul adresează serverului o cerere prin care solicită un anumit serviciu pe care trebuie să îl execute serverul.

Se poate spune că o aplicație client-server include problema emițător-receptor din următorul punct de vedere: atât în momentul în care clientul adresează serverului cererea de a îi oferi un serviciu, cât și atunci când serverul transmite clientului său rezultatul acțiunii solicitate, are loc o comunicare de tip emițător-receptor (sau cazuri particulare ale acesteia).

Din punct de vedere practic, cea mai la îndemână aplicație client-server stă la baza utilizării *Internet*-ului. În 1990 ***Tim Berners-Lee*** a dezvoltat un sistem standard de tip *hipermedia*, numit ***World Wide Web***, pe scurt, ***WWW***. Domeniul de aplicare inițial a fost fizica nucleară, la centrul CERN din Elveția. Esența sistemului consta în faptul că nucleul său îl reprezenta partea de comunicare în rețea și astfel se crease posibilitatea transferului în sistem a paginilor hipermedia, fără restricții.

În 1993 centrul *NCSA* (*National Center for Supercomputing*), Universitatea din *Illinois*, a dezvoltat browserul *Mosaic*. Importanța și noutatea acestei aplicații era interfața comodă, orientată pe ferestre, care recunoștea transferul paginilor *hipermedia* într-un sistem de informare. Începând cu *Mosaic*, amploarea pe care au luat-o sistemele de tip *WWW* a fost explozivă.

Enorm de multe companii, universități și particulari și-au construit servere de informare care să poată fi citite (vizitate) de oricine posedă un browser.

Fiecare pagină *hipermedia* poate să conțină text, imagini și legături către alte pagini. O astfel de pagină este creată folosind un limbaj de marcare de tip *HTML* (*HyperText Markup Language*), care este similar cu limbajul standard *SGML*. În plus față de *SGML*, limbajul *HTML* are pune la dispoziție instrucțiuni referitoare la modul de afișare a paginii create.

Într-un server de informare cum este *WWW* fiecare pagină este identificată printr-un nume, adică o adresă *URL* (*Universal Resource Locator*), care se folosește ca legătură către pagina respectivă.

Dacă o pagină *P-părinte* trebuie să vadă (să conțină o legătură către) o altă pagină, *P-copil*, atunci în pagina *P-părinte* afișată vom avea un text de legătură (*hiperlegătură*) către pagina *P-copil*.

În același timp, codul *HTML* al paginii *P-părinte* va conține atât textul de legătură cât și adresa *URL* a paginii *P-copil*. Deci, în pagina părinte nu apare obligatoriu adresa *URL* a paginii copil. Pentru a deschide pagina copil, este suficient un click de mouse pe textul de legătură din pagina părinte. Pentru a localiza pagina copil, browserul contactează serverul care conține pagina identificată de legătura aleasă pe pagina părinte, transferă informația furnizată de acel server și o afișează în locul paginii părinte.

Protocolul de comunicare respectat de browser și de server este numit *HTTP* (*HyperText Transfer Protocol*).

Fiecare client (browser) lucrează secvențial, putând să vizualizeze (să afișeze) la un moment dat o singură pagină. Un server oarecare poate să primească concurent oricâte cereri de furnizare de pagini de informații. Deci avem aici un exemplu în care serverul trebuie să deservească simultan, concurent, mai mulți clienți.

O extindere recentă în tehnologia paginilor *Web* este adăugarea de secvențe animate pe pagini, scrise de obicei în limbajul de programare *Java*.

Aici prelucrarea concurentă poate fi realizată la nivelul clientului pentru a mări viteza de animare și pentru a permite animației să co-existe pe pagină în paralel cu acțiunile utilizatorului, de exemplu cu perioada de încărcare a unei forme pe pagina curentă.

Problema bancomatului

Această problemă stă la baza gestionării operațiilor de utilizare a bancomatelor aceleiași bănci de către diferiți utilizatori, simultan și concurent. Enunțul problemei este următorul:

Un client al unei bănci, identificat printr-un număr personal, încearcă să scoată o anumită sumă de bani de la un bancomat. În acest scop, clientul va trebui să introducă mai întâi numărul personal, pe care automatul îl verifică. Dacă numărul personal este corect atunci clientul este solicitat de către automat să introducă suma de bani pe care dorește să o scoată. Dacă, în plus, această sumă solicitată nu depășește valoarea totală a contului atunci clientului i se eliberează de către automat suma dorită. Se cere să se simuleze aceste activități.

Caracterul paralel al problemei este imediat: bancomate diferite pot derula în același timp activități independente. Caracterul concurent al acestei probleme rezultă din faptul că soluția corectă trebuie să rezolve situația în care mai mulți clienți solicită simultan accesul la aceeași bază de date a conturilor aceleiași bănci. Sau, mai mult, clienți diferiți solicită în același timp, de la bancomate diferite, sume de bani din același cont!

Se vede că problema este în strânsă legătură cu teoria bazelor de date, chiar a bazelor de date concurente, în care conceptul de tranzacție este fundamental. Exploatând facilitățile oferite de sistemele de gestiune a bazelor de date pentru asigurarea consistenței bazelor de date la derularea tranzacțiilor, programatorul poate rezolva de o manieră proprie, convenabilă, problema bancomatului, conform cu restricțiile pe care dorește să le impună.

În loc de încheiere, două invitații...

Pentru vizualizarea acestor probleme vă recomandăm să vizitați următoarele pagini *Web*:

- www.csd.uch.gr/~hy345/mirror/workbench/centsync.html
- www.hta-be.bfh.ch/~fischli/kurse/threads
- www.cs.buffalo.edu/~bina/cse421/fall99/lectures/lec5/index.htm.

De asemenea, pentru ca serialul acesta să vă fie cât mai util, nu ezitați să ne contactați la vasilex@info.unitbv.ro. Vă așteptăm cu real interes!

Bibliografie

[1] Rancea Doina, *Limbajul Turbo Pascal*, Editura Libris, Cluj-Napoca, 1994, vol. 1

[2] Georgescu Horia, *Programare concurentă - teorie și aplicații*, Editura Tehnică, București, 1994

[3] Cocan Moise, Vasilescu Anca, *Programarea matematică folosind MS-Excel, Management Scientist, Matlab*, Editura Albastră, Cluj-Napoca, 2000

[4] Boian Florian Mircea, *Programarea distribuită în Internet - metode și aplicații*, Editura Albastră, Cluj-Napoca, 2000

*** *Programare concurentă Cover Story*, PC Report 40 (ianuarie 1996)

*** *Procesarea distribuită Cover Story*, PC Report 50 (noiembrie 1996)

*** *Modelul client/server Cover Story*, PC Report 54 (martie 1997)

O posibilă definire ALGEBRICĂ a tipurilor ORDINALE

Tiberiu Socaciu

Vom încerca în cele ce urmează să realizăm o introducere algebrică a tipurilor de date ordinale din limbajul Pascal. Aparatul algebric folosit în articol este prezentat pe scurt la început. Considerăm că el este usor comprehensibil de către orice elev cu abilităti matematice peste medie.

Vom începe cu prezentarea unor definiții algebrice pe care le vom folosi în prezentarea unor explicații ulterioare.

Produs cartezian
Numim *produs cartezian* a două mulțimi A și B mulțimea $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$.

Relație
Numim *relație* peste o mulțime M un triplet r de forma $r = (M, M, R)$, unde $R \subseteq M \times M$.

Notații
Pentru o relatie $r = (M, M, R)$, vom considera următoarele notații echivalente:
- $(x, y) \in R$
- $x\ r\ y$.

Relație reflexivă
Spunem că o relație $r = (M, M, R)$ este *reflexivă* dacă
$$\forall x \in M: (x, x) \in R.$$

Relație antisimetrică
Spunem că o relație $r = (M, M, R)$ este *antisimetrică* dacă
$$\forall x, y \in M: (x, y) \in R \wedge (y, x) \in R \rightarrow x = y.$$

Relație tranzitivă
Spunem că o relație $r = (M, M, R)$ este *tranzitivă* dacă
$$\forall x, y, z \in M: (x, y) \in R \wedge (y, z) \in R \rightarrow (x, z) \in R.$$

Relație de ordine
Spunem că o relație r peste o mulțime M este o *relație de ordine* dacă îndeplinește simultan condițiile:
- r este reflexivă;
- r este antisimetrică;
- r este tranzitivă.

Relație totală
Spunem că o relație r peste o mulțime M este o *relație totală* dacă $\forall x, y \in M: (x, y) \in R \vee (y, x) \in R$.

Mulțime de ordine
Numim *mulțime ordonată* (sau *lanț*) o mulțime înzestrată cu o relație de ordine totală.

Morfism de ordine
Numim *morfism de ordine* de la mulțimea ordonată (M_1, r_1) la mulțimea ordonată (M_2, r_2) o funcție $f: M_1 \rightarrow M_2$ pentru care
$$\forall x, y \in M_1: x\ r_1\ y \rightarrow f(x)\ r_2\ f(y).$$

Izomorfism de ordine
Numim *izomorfism de ordine* un morfism de ordine care este bijectiv.

Mulțimi izomorfe
Spunem că două mulțimi ordonate sunt *izomorfe* dacă există între ele un izomorfism de ordine.

Inversa unei relații

Inversa unei relații $r = (M, M, R)$ se definește ca fiind $r^{-1} = (M, M, R^{-1})$, unde

$$(x, y) \in R^{-1} \rightarrow (y, x) \in R.$$

Relație diagonală

Relația diagonală (sau *relația de egalitate*) peste o mulțime M se definește ca fiind $= (M, M, A)$, unde

$$A = \{(x, x) \mid x \in M\}.$$

Negația unei relații

Negația unei relații $r = (M, M, R)$ se definește ca fiind

$$\neg r = (M, M, M \times M / R).$$

Intersecția a două relații

Având două relații $r_1 = (M, M, R_1)$ și $r_2 = (M, M, R_2)$ se definește *intersecția* lor ca fiind relația

$$r_1 \cap r_2 = (M, M, R_1 \cap R_2).$$

Mulțimea părților

Mulțimea părților unei mulțimi A se definește ca fiind mulțimea $\wp(A) = \{B \mid B \subseteq A\}$

Funcția maxim

Fie A o mulțime finită ordonată și B o submulțime nevidă a sa. Funcția $MAX: \wp(A) \setminus \{\emptyset\} \rightarrow A$, astfel încât

$$\forall B \in \wp(A) \setminus \{\emptyset\}: MAX(B) \in B$$

și

$$\forall x \in B: x \leq MAX(B)$$

se numește *funcția maxim*, iar $MAX(B)$ se numește *maximul* mulțimii B.

Funcția minim

Funcția minim se definește analog: $MIN: \wp(A) \setminus \{\emptyset\} \rightarrow A$, astfel încât

$$\forall B \in \wp(A) \setminus \{\emptyset\}: MIN(B) \in B$$

și

$$\forall x \in B: x \geq MIN(B)$$

Mulțimile succesorilor și predecesorilor

Fia A o mulțime ordonată. Spunem că mulțimile $S(x)$ și $P(x)$ sunt *mulțimile succesorilor* lui x, respectiv ale *predecesorilor* lui x, unde

$$S(x) = \{y \in A \mid x \leq y, x \neq y\}$$

și

$$P(x) = \{y \in A \mid x \geq y, x \neq y\}.$$

Tipuri de date

Se știe că una din posibilele definiții ale tipului de date este cea a unui triplet de forma:

(*Mulțime_de_valori*, *Operații_și_relații*, *Reprezentarea_valorilor_în_memorie*).

Practic, dacă facem abstracție de cea de-a treia componentă, tipurile de date pot fi privite ca structuri algebrice (adică mulțimi suport înzestrate cu o mulțime de operații cu diferite arități).

Tipuri ordinale

Tip ordinal

Numim tip *ordinal* un lanț izomorf cu o mulțime finită de numere întregi.

Propoziție

Fie T un tip ordinal. Atunci:

- T este finit;
- $\exists k \in N \setminus \{0\}, \exists ORD: T \rightarrow \{0, 1, ..., k - 1\}$ un izomorfism de ordine.

Exercițiu

Demonstrați propoziția de mai sus.

Observație

Funcția *ORD* este predefinită în *Pascal*.

Observație

Relația de ordine totală "≤" este predefinită în *Pascal* și se notează cu "<=".

Notații

- Inversa relației "<=" este notată în *Pascal* cu ">=".
- Relația diagonală se notează în *Pascal* cu "=".
- Negația relației "=" se notează în *Pascal* cu "<>".

Observație

Relațiile definite anterior sunt predefinite în *Pascal*.

Relații de ordine strictă

Definim două *relații de ordine strictă* astfel:

- "<" := "<=" ∩ "<>";
- ">" := ">=" ∩ "<>".

Observație

Relațiile definite anterior sunt predefinite în *Pascal*.

Funcțiile succesor și predecesor

Funcțiile

$$SUCC: T \setminus MAX(T) \rightarrow T \setminus MIN(T)$$

și

$$SUCC: T \setminus MAX(T) \rightarrow T \setminus MIN(T)$$

date de

$$SUCC(x) = MIN(S(x))$$

și

$$PRED(x) = MAX(P(x))$$

poartă numele de funcția *succesor* și de funcția *predecesor*.

Observație

Funcțiile prezentate sunt predefinite în limbajul *Pascal*.

Propoziție

Pentru orice x pentru care funcțiile sunt definite, avem:

- $SUCC(x) > x$;

- $PRED(x) < x$;
- $SUCC(PRED(x)) = x$;
- $PRED(SUCC(x)) = x$;
- $ORD(SUCC(x)) = ORD(x) + 1$;
- $ORD(PRED(x)) = ORD(x) - 1$.

Exercițiu

Demonstrați propoziția de mai sus. Care sunt condițiile pe care trebuie să le respecte argumentul x în fiecare din cele șase cazuri?

Tipurile ordinale ale limbajului Pascal

Tipuri enumerare

În cazul tipurilor enumerare, lanțul se definește explicit prin intermediul unei liste de identificatori:

$$(e_1, e_2, ..., e_k),$$

unde

$$ORD\ (e_i) = i - 1.$$

Tipuri subdomeniu

În cazul tipurilor subdomeniu, lanțul se definește implicit prin intermediul capetelor subdomeniului:

$$a..b,$$

unde a și b au același tip ordinal A. În acest caz elementele mulțimii sunt:

$$\{x \in A \mid a <= x \wedge x <= b\},$$

iar

$$ORD(x) = ORD_A(x) - ORD_A(a).$$

Tipul Boolean

Tipul `Boolean` este un tip enumerare predefinit:

```
Boolean=(false,true).
```

Tipul întreg

Tipul întreg este un tip subdomeniu predefinit:

```
Integer = -Maxint..Maxint,
```

unde `Maxint` este o constantă dependentă de implementare.

Observații

- Trebuie remarcat faptul că în cazul tipului întreg, funcția *ORD* este funcție identică pe `Integer`, fiind un caz în care nu se respectă, din rațiuni de implementare, regulile stabilite.
- Limita inferioară pentru `Integer` are, în cele mai multe din cazuri, valoarea `-Maxint - 1`, datorită implementării folosind virgula fixă.

Tipul caracter

Tipul caracter este un tip subdomeniu predefinit:

```
Char = CHR(0)..CHR(255),
```

unde funcția

```
CHR:0..255 → Char
```

este un izomorfism de ordine, a cărui inversă este funcția *ORD*.

Alte tipuri întregi

În implementările *Turbo/Borland Pascal* ale firmei *Borland* au fost definite și alte tipuri întregi: `Shortint`, `Byte`, `Word`, `Longint` care sunt compatibile (în sensul atribuirii) cu tipul întreg.

Exercițiu

Celelalte tipuri intregi (altele decât `Integer`) sunt sau nu ordinale? Justificați răspunsul.

Dl. Tiberiu Socaciu este licențiat și master al Facultății de Matematică, Universitatea "Babeș-Bolyai" Cluj. Este director IT la SC InfoData SRL Cluj, și poate fi contactat prin e-mail la adresa **socaciu@inf.ro**.

www.price.ro

Site-ul **www.price.ro** este un ghid online de prețuri pentru componente, periferice, accesorii și sisteme PC.

Vă este pusă la dispoziție o bază de date care conține ofertele celor mai importanți distribuitori din România.

Informațiile referitoare la produse cuprind denumirea, specificațiile, prețul, producătorul și firma distribuitoare. Pentru fiecare distribuitor vă este pusă la dispoziție o pagină care conține modalitățile de contact și data la care a fost actualizată oferta acestuia.

Interfața este foarte prietenoasă, informațiile putând fi găsite foarte ușor.

Pentru a căuta un anumit produs va trebui să selectați mai întâi categoria din care acesta face parte (componente, periferice, accesorii etc.), și apoi familia din care face parte. De exemplu, dacă doriți să căutați un procesor veți selecta meniul **Componente** și apoi **Procesoare**. În acest caz veți putea alege prețul maxim, frecvența minimă, frecvența magistralei și platforma. Va fi afișată o listă ce conține informațiile care respectă cerințele specificate.

Aveți la dispoziție și un **Wish Basket** care va conține produsele pe care le doriți (și prețul total), precum și posibilitatea de a citi sau de a adăuga anunțuri de vânzare/cumpărare.

La cules de... ALGORITMI

Sorin Groza, Ionuț Stan

Sosirea toamnei înseamnă belșug în hambare, dar totodată și începerea unui nou an școlar. Suntem convinși că, pentru toți cei care iubesc informatica și în special algoritmii, aceasta reprezintă un bun prilej în vederea îmbogățirii cunoștințelor necesare în vederea participării la concursurile viitoare.

După o lungă vară fierbinte vă propunem câteva probleme "răcoritoare". Vă asigurăm că vom mai "coace", chiar dacă vine toamna, noi probleme, la care să găsiți algoritmi cât mai interesanți.

Matematica... pe calculator

La lecția de matematică profesorul le cere elevilor să efectueze o înmulțire a unor numere al căror produs depășește "limitele matematicii". Primul care va rezolva această problemă va avea media 10 în acel semestru. Unul dintre ei, fiind mai isteț și având lângă el un calculator de buzunar, a început să efectueze înmulțirile pe calculator, dar a observat că în acest caz nici calculatorul de buzunar nu îi este de folos. Dar dacă tot este el mai isteț a apelat la informaticieni și a spus: "Primul care îmi va rezolva problema va primi temele corecte la matematică în tot acest semestru".

Cerință

Se citesc n numere întregi din fișierul PRODUS.IN. Se cere să se determine produsul acestor numere. Soluția se va scrie în fișierul PRODUS.OUT.

Restricții

- $2 \le n \le 20$
- $-25000 \le x_i \le 25000$

Exemplu

PRODUS.IN
3
1234
-12345
2457

PRODUS.OUT
-37429274610

Numere apropiate

Se dă un număr natural n (care conține cel mult 2457 de cifre). Să se determine cel mai mic număr natural $m \ge n$ care acceptă ca divizor un număr dat d ($1 \le d \le 2457$).

Numerele n și d se citesc de pe primele două linii ale fișierului NUMERE.IN, iar numărul m va fi scris pe prima linie a fișierului NUMERE.OUT.

Exemplu

NUMERE.IN
14
5

NUMERE.OUT
15

La pescuit

Elevii informaticieni, având prea multe concursuri de informatică, s-au gândit că ar fi bine să aibă o perioadă de recreere și au organizat un concurs de pescuit la care au participat foarte mulți concurenți. Fiecare elev a prins pești de diferite specii, speciile fiind identificate prin numere. Numărul peștilor prinși este același pentru fiecare elev. Se spune că între doi elevi există relație de legatură dacă ei au prins cel puțin un pește de aceeași specie, astfel formându-se grupuri. Să se determine numărul de grupuri care se vor forma și să se afișeze membrii fiecărui grup.

Datele de intrare se citesc din fișierul PESTI.IN care are următorul format:

- pe prima linie numărul n al elevilor;
- pe a doua linie numărul k al peștilor prinși de fiecare elev;
- pe fiecare dintre următoarele n linii câte k numere care indică specia fiecărui pește prins de un anumit elev.

Datele de ieșire se scriu în fișierul PESTI.OUT care are următorul format:

- pe prima linie numărul *g* al grupurilor formate;
- pe fiecare dintre următoarele *g* linii elevii dintr-un grup.

Exemplu
PESTI.IN
10
3
1 2 1
1 2 2
3 3 3
3 4 5
7 8 7
7 8 7
8 8 7
9 9 9
1 1 2
7 7 8

PESTI.OUT
4
1 2 9
3 4
5 6 7 10
8

La scăldat

La ştrandul din oraş s-a organizat un concurs de înot dotat cu premii. Din dorinţa de a câştiga marele premiu s-au înscris şi copii care nu ştiau să înoate. Pentru siguranţa lor ei nu au fost acceptaţi în concurs. Deoarece concurenţii aveau vârste diferite, s-a hotărât ca la timpul fiecărui concurent (măsurat în secunde) să se adune vârsta acestuia. Deoarece a început ploaia, s-a anunţat că se oferă un abonament celui care scrie un program care va stabili ordinea finală, pentru ca premierea să aibă loc cât mai repede.

Fişierul de intrare STRAND.IN are următorul format:
- pe prima linie numărul *n* al copiilor înscrişi; copiii sunt identificaţi cu numere cuprinse între 1 şi *n*.
- pe linia a doua numărul *k* al copiilor eliminaţi;
- pe linia a treia *k* numere indicând copiii eliminaţi;
- pe linia a patra *n* - *k* numere indicând timpii celor rămaşi în concurs, în ordinea numerelor primite la înscriere;
- pe linia a cincia *n* - *k* numere indicând vârstele celor rămaşi în concurs, în ordinea numerelor primite la înscriere.

În fişierul de ieşire STRAND.OUT se va afişa ordinea finală, fiecărei poziţii corespunzându-i o linie. În caz de egalitate între doi sau mai mulţi concurenţi, ei vor apărea pe aceeaşi linie.

Exemplu
STRAND.IN
7
2
1 3
60 55 58 55 70
8 14 10 10 9

STRAND.OUT
6
2 5
4
7

Drumul spre şcoală

Gigel este un elev căruia nu-i prea place şcoala şi ar face orice să nu mai ajungă acolo. În a 43-a zi a anului şcolar mama sa l-a trimis la şcoală la ora *P* şi *P'* minute. El trebuia să ajungă la şcoală la ora *PP* şi *PP'* minute.

În drum spre şcoală el se întâlneşte cu câţiva prieteni care îl ţin de vorbă de la ora *D* şi *D'* minute până la ora *DD* şi *DD'* minute. Dacă discută mai mult de 43 de minute, atunci ei vor mai discuta încă 43 de minute.

Scrieţi un program care verifică dacă Billy ajunge la timp la şcoală şi în cazul în care ajunge, să se determine ora şi minutul la care a ajuns.

Durata deplasării de acasă până la şcoală este de 43 de minute.

Intrare
Datele de intrare se citesc din fişierul GIGEL.IN care are următorul format:
- pe prima linie se află ora *P* şi minutul *P'* la care Gigel pleacă de acasă;
- pe a doua linie se află numărul *N* al prietenilor cu care se întâlneşte Gigel pe drum;
- fiecare dintre următoarele *N* linii conţin ora *D* şi minutul *D'* de întâlnire şi ora *DD* şi minutul *DD'* la care se termină conversaţia;
- pe ultima linie este scrisă ora *PP* şi minutul *PP'* la care trebuie să ajungă Gigel la şcoală.

Ieşire
Rezultatul se va tipări la ieşirea standard. Dacă Gigel ajunge la şcoală se va tipări cuvântul DA, iar dacă nu, se va tipări cuvântul NU. În cazul în care Gigel reuşeşte să ajungă la şcoală se va mai tipări ora şi minutul la care ajunge.

Exemplu
GIGEL.IN
7 30
2
7 32 7 34
7 36 7 37
8 43

Ecran
DA
8 16

Dl. prof. Sorin Groza este cadru didactic la Şcoala Generală "Ovidiu Hulea" din Aiud, iar Ionuţ Stan este elev la aceeaşi şcoală. Pot fi contactaţi prin intermediul redacţiei.

În numărul URMĂTOR

La mijlocul lunii noiembrie va apărea următorul număr al revistei noastre. Vă prezentăm pe scurt câteva dintre articolele pe care le veți putea citi...

Microsoft Windows XP

Data oficială a lansării noului sistem de operare al companiei Microsoft este 25 octombrie 2001. Veți putea citi o scurtă prezentare a noilor caracteristici introduse de această versiune, printre care și noua interfață grafică proiectată în așa fel încât să elimine o parte dintre inconvenientele cu care se confruntau utilizatorii versiunilor anterioare.

AMD Athlon XP

Pe data de 9 octombrie a fost lansată pe piață versiunea pentru desktop-uri a procesorului cunoscut până acum sub numele de cod Palomino. Deoarece data lansării este foarte apropiată de cea a lansării sistemului de operare Microsoft Windows XP, cei de la AMD au adăugat aceeași terminație pentru denumirea noului procesor pentru a putea profita din plin de oportunitatea comercială oferită de această coincidență (chiar dacă motivele oficiale sunt altele și semnificația acestei terminații este diferită). În următorul număr al revistei vom prezenta câteva dintre caracteristicile celui mai performant procesor oferit în acest moment de compania americană AMD.

Fractali

Teoria fractalilor și cea a haosului se numără printre cele mai intersante și mai noi ramuri ale matematicii. Cu o istorie relativ recentă (aproximativ 50-60 de ani) ele s-au impus repede în lumea științei având aplicații în aproape toate domeniile, de la informatică până la geografie, economie, statistică etc. În acest articol veți putea citi o introducere în teoria fractalilor și a haosului; aceasta este urmată de o prezentare a celor mai celebri fractali, printre care se regăsesc triunghiul lui Sierpinski, garnitura lui Sierpinski, covorul lui Sierpinski, curba lui Peano de umplere a spațiului, seriile Mandelbrot și seriile Julia. Va fi explicată noțiunea de dimensiune fractală și vor apărea definițiile unor termeni des folosiți.

Rețele neuronale

Unul dintre cele mai interesante domenii ale informaticii este, fără îndoială, inteligența artificială (IA). Progresele pe care acest domeniu le-a înregistrat în ultimul timp sunt demne de luat în seamă, dar scopul final (crearea unui dispozitiv capabil să raționeze) este încă departe de a fi atins. În cadrul IA, o ramură foarte importantă o constituie studiul rețelelor neuronale. În numărul următor veți putea citi un articol care prezintă câteva detalii referitoare la modul în care se poate încerca simularea comportamentului unui neuron folosind uneltele pe care ni le pun la dispoziție calculatoarele de astăzi.

NETREPORT

PC REPORT ÎN MILENIUL III

www.pcreport.ro

Conectat la viitorul digital

AGORA MEDIA Str. Tudor Vladimirescu nr. 63/9, Tg. Mureş, CP 230/1, 4300

Tel: 065-166 516, Fax: 065-166 290, e-mail: office@agora.ro

AGORA MEDIA

OFERTĂ SPECIALĂ!

Pentru abonamentele încheiate la revista

PC Magazine România

în perioada 22 octombrie–31 decembrie

beneficiați de o

REDUCERE DE 52% !

Preț vânzare 12 numere ~~414.000~~

Abonament 12 numere* **200.000 lei**

www.pcmagazine.ro

* Cei care sunt abonați deja, pot folosi oferta pentru a-și prelungi abonamentul

Ghidul utilizatorilor de tehnologii informatice și comunicații

AGORA MEDIA S.R.L. str. C-tin Rădulescu Motru nr. 13, etaj 4, apt. 42
sect. 4 (CP 94/49), București, Tel: 01-3309282, Fax: 01-3309285

gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

Noiembrie 2001
Vol. 11/7

Lei 20.000

din sumarul numărului:

Microsoft® WindowsXP

Civilization III

Soluții

Olimpiada Internațională de Informatică 2001

AMD Athlon XP

Considerații asupra fractalilor
Realizarea interfețelor conversaționale
Bursele Agora

AGORA

pe web

www.agora.ro

www.pcmagazine.ro

www.netreport.ro

www.eweek.ro

www.pcconcrete.ro

news.agora.ro.

www.ginfo.ro

www.aol.ro

conectează-te
ACUM!

Vol. 11, Nr. 7 /2001

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Redactor:
Carmen Șușman (Alba Iulia)
Au mai colaborat:
Lucian Burja (Blaj)
Ciprian Ileană (Alba Iulia)
Ioana-Maria Ileană (Alba Iulia)
Mihai Oltean (Cugir)
Mihai Stroe (București)
Ion-Gelu Tomșa (Dej)
Tehnoredactare:
Claudiu Soroiu
Coperta și grafica:
Andrei Pașa
Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 064/192.422
065/166.516
Fax: 065/166.290
http://www.ginfo.ro

ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359
Editor: Agora Media SRL
Editor al publicațiilor "eWeek România", "PC Magazine România", "NetReport" și "Gazeta de Informatică"
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihai Bucuroiu
Bogdan Manu
Florin Tiță
Denis Marioara
Marius Comărniceanu
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

Sumar

actual.. PAG. 5

Produsele lunii
Athlon XP
Windows XP
Civilization III
REFERATE

probleme.. PAG. 9

propuse pentru REZOLVARE

Olimpiada de Informatică a Europei Centrale

~Zalaegerszeg, Ungaria~

soluții.. PAG. 15

Olimpiada Internațională de Informatică

~Tampere, Finlanda~

concurs.. PAG. 21

Etapa finală a ediției anterioare

Talon de participare

Problemele de la a doua rundă a noii ediții

focus.. PAG. 31

PERCEPTRONUL cu un singur strat

Lucian BURJA

~rețele neuronale~

babel.. PAG. 35

Considerații asupra FRACTALILOR

Cirpian ILEANĂ
Ioana-Maria ILEANĂ

~ordinea haosului~

internet.. PAG. 44

Realizarea interfețelor CONVERSAȚIONALE

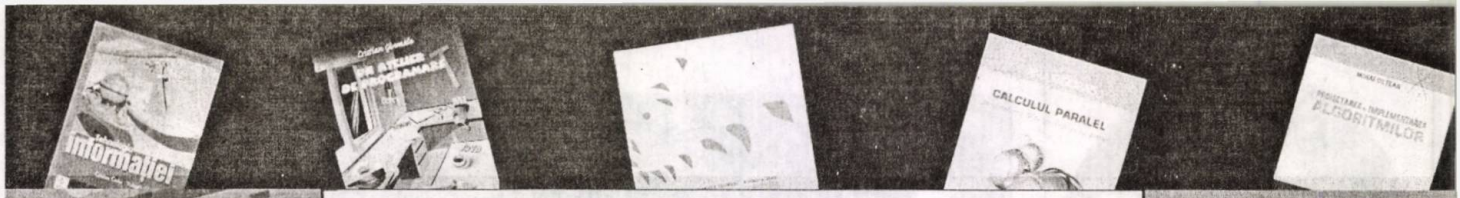
Ion-Gelu TOMȘA

~Microsoft Agent~

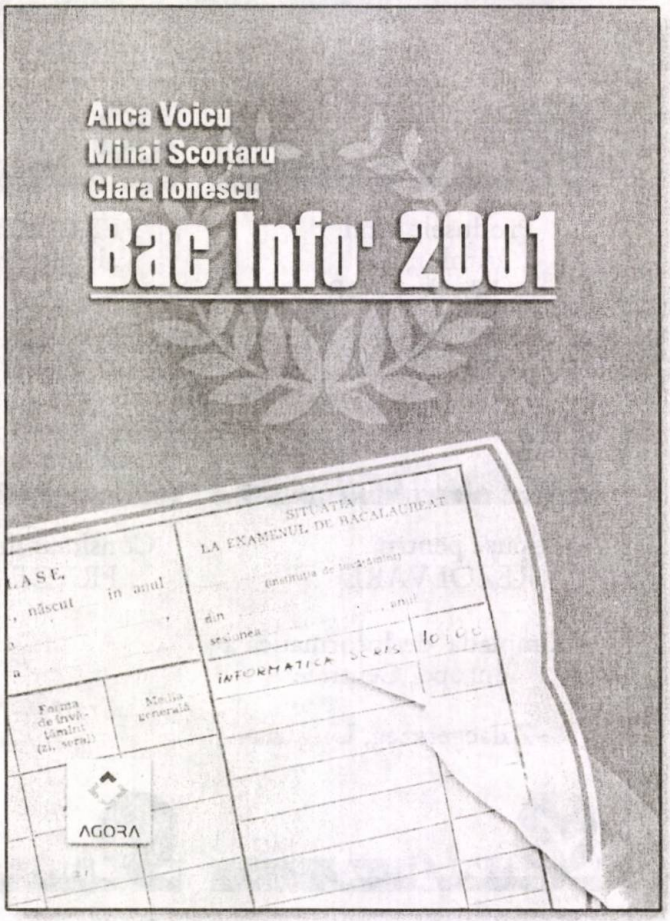
dialog.. PAG. 48

În numărul URMĂTOR

scurtă prezentare a articolelor din numărul 11/8 al GInfo



COMPUTER LIBRIS AGORA



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Limba ROMÂNĂ

Mulți elevi pasionați de informatică cred că orele de limba și literatura română sunt o pierdere de vreme. De multe ori întreabă, retoric, la ce le poate folosi în viitor faptul că știu să comenteze operele lui Eminescu, Sadoveanu etc. Evident, ei se lasă greu convinși...

Exprimarea

Datorită faptului că informatica devine tot mai atractivă, promite un viitor sigur și incitant pentru cei care știu să utilizeze calculatorul în diverse scopuri, în ultima vreme cititorii revistei noastre trimit multe materiale redacției cu dorința de a le publica. Dar, destul de frecvent, când primim câte un articol ne întrebăm cum de reușesc autorii noștri să comită atât de multe greșeli de exprimare... și nu numai. Urmează ca în redacție să transformăm textul deficitar, să corectăm exprimarea, apoi să trimitem autorului articolul cu rugămintea de a-l revizui. Nu de puține ori, autorii au preferat să ne lase pe noi să decidem (în locul lor) care este forma potrivită a articolului care se va publica. Este adevărat că sarcina noastră este să publicăm în revistă articole corecte atât din punct de vedere științific cât și al exprimării, al ortografiei etc.; dar nu putem să nu vă atragem atenția că articolul va fi semnat totuși de autor...

Din păcate mulți consideră că exprimarea are un rol secundar, că este mult mai important conținutul științific, iar cititorii vor înțelege oricum. Este evident că o publicație care se respectă cât de cât nu poate prelua pentru publicare un text nefinalizat din punct de vedere stilistic. Nu credem că ați dori să citiți anumite articole în forma în care au fost trimise de autorii lor.

Au fost cazuri în care autorul, după ce și-a văzut articolul publicat, ne-a acuzat că l-am "măcelărit", că nici măcar nu mai seamănă cu ceea ce a trimis el. Simțim că este nevoie de un exemplu; ce ziceți de o propoziție de genul: "*Expresia minimă sumă de produse pentru o funcție constă dintr-o sumă de anumiți implicanți primi ai funcției*"?

Autorului i s-a părut că totul este foarte clar și s-a mirat cum de nu înțelegem ce a vrut să spună. Într-o eventuală polemică, i-am putea da replica: "*un oraș ește intersectat de străzi care se intersectează cu un polițist în sens unic și dublu.*"

De această dată este mult mai probabil ca interlocutorul să declare că nu înțelege semnificația propoziției. Explicația este foarte simplă: prima propoziție îi aparține, în timp ce a doua a fost scrisă de un alt autor.

Aceste situații apar datorită faptului că majoritatea informaticienilor consideră că dacă ei înțeleg sensul unei propoziții, atunci oricine ar trebui să îl înțeleagă. Aceștia nu își dau seama că cei care citesc propoziția nu au cum să știe ce au ***vrut*** ei să spună. Este puțin probabil ca cititorii să "ghicească" acele idei care există frumos rânduite în mintea autorului, dar care nu au fost exprimate la fel de clar.

Problema fundamentală constă în atrofierea talentului unora dintre noi de a vorbi corect și clar atunci când ar trebui să ne explicăm ideile, să le exprimăm în limba română astfel încât să ne facem înțeleși.

Limba română

Ar trebuie ca toți să ne dăm seama de rolul pe care l-au avut și îl au orele de limba și literatura română.

Este clar că acestea ne dezvoltă vocabularul, priceperea de a exprima aceeași idee în mai multe feluri. De formarea acestor deprinderi ar trebui să se preocupe nu doar profesorul de limba și literatura română, cât și cel de informatică... Cu siguranță ați auzit de profesori cu titlu de doctor care nu reușesc să explice cunoștințe, de altfel banale, astfel încât elevii sau studenții să-i înțeleagă. Pe de altă parte, ce părere ați avea despre profesorul vostru dacă acesta ar răspunde la o anumită întrebare, repetată eventual de mai mulți elevi, cu aceleași cuvinte pe care nici prima dată nu le-ați înțeles?

Prescurtări

Mulți dintre cei care își petrec o parte a timpului liber folosind servicii *IRC* consideră că este suficient ca doar cei cu care vorbesc să înțeleagă ce spun ei. La un moment dat, din curiozitate, am încercat să folosesc acest tip de servicii. Am renunțat foarte repede deoarece nu înțelegeam ce vor să spună interlocutorii. Se pare că mulți au uitat că în limba română există și vocale. Pentru a comunica mai repede se folosesc o mulțime de prescurtări pe care un nou venit nu are cum să le înțeleagă. Până la urmă se ajunge în situația în care sunt create prescurtări de dragul prescurtărilor. Apar situații hilare în care "prescurtările" unor termeni sunt mai lungi decât termenii respectivi.

Libertatea de care au parte cei care conversează pe *Internet* este reală și din păcate cei care beneficiază de ea o folosesc în scopul creării unei lumi ireale, virtuale...

Semne de punctuație

O problemă delicată este cea a semnelor de punctuație. Există câteva reguli simple care ar trebui respectate:

- cuvintele trebuie separate printr-un singur spațiu;
- semnele de punctuație (, . : ; ! ?) trebuie "lipite" de cuvintele care le preced;
- parantezele deschise și ghilimelele de început trebuie "lipite" de cuvintele pe care le preced, iar parantezele închise și ghilimelele de sfârșit urmează imediat după cuvintele care le preced, fără a mai apărea spații în fața lor.

Chiar dacă regulile sunt foarte simple, ele nu sunt respectate. Nerespectarea lor nu îi deranjează pe cei care scriu, în schimb nu le place când le văd. De exemplu, nu credem că v-ar plăcea să găsiți în *GInfo* texte de genul: " *Apariția noului procesor cu frecvența de 1,7 GHz , alături de reducerile generale de prețuri practicate de Intel ,a dus la scăderea prețurilor produselor din această familie . Reducerile nu se vor opri aici;prețurile vor scădea și mai mult în cursul acestei veri* ."

Romgleza

Este la modă folosirea termenilor preluați din limba engleză. În anumite conjuncturi este normal să fie așa, mai ales atunci când nu există traduceri acceptabile pentru unii termeni.

Am putea să înșirăm o serie de asemenea cuvinte, preluate pe neobservate în limba română. Cu toții ne-am obișnuit cu *hardware* și *software*, cu *link* și *pointer* sau cu *heap* și *mouse*. Nu s-au tradus, sau, mai bine zis, traducerea nu a intrat în uz pentru că nu sugerează perfect sensul dorit. Degeaba încercăm să folosim "legătură" în loc de *link*, degeaba căutăm un cuvânt românesc pentru *mouse*.

Acum nu despre aceste *romglezisme* vorbim, ci despre amestecarea nejustificată a celor două limbi. De exemplu, ar fi interesant ca în *GInfo* să apară următoarea frază: "*If iei un număr, determinarea primalității sale is very simplă. Important e to know algoritmul. Pentru cine don't știe, nu are decât to learn doing altceva*".

Gramatica

Constatăm frecvent că unele articole conțin o mulțime de greșeli gramaticale. Nu sunt rare propozițiile de genul: "*Pentru a determina rădăcina arborelui, algoritmul trebuiește să parcurgă toate nivelele acestuia.*"

Faptul că verbul "a trebui" este impersonal este cunoscut de majoritatea cititorilor. Din păcate, acest lucru este ignorat de prea multe ori.

Mai puțini știu că pluralul corect al substantivului "nivel" este "niveluri". "Nivele" are forma de singular "nivelă" și are o semnificație cu totul diferită. Această eroare apare în paginile majorității revistelor de IT din *România*. Probabil că această "scăpare" nici nu mai este considerată greșeală, deoarece majoritatea cititorilor s-au obișnuit cu ea.

Situația este diferită în cazul articolelor genitivale. Dezacorduri de genul "*numărul al cărei rădăcină pătrată*" nu pot fi acceptate.

Exemplele ar putea continua, dar nu vrem să transformăm acest editorial într-o prelegere privind corectitudinea gramaticală.

Sfaturi

Nu vrem să vă plictisim cu fraze de genul: "*trebuie să vorbiți și să scrieți corect românește*". În majoritatea situațiilor, greșelile de limbă (română, engleză, franceză etc.) "pot fi ignorate", în special în timpul unei conversații.

Dar nu fiți nicicând neglijenți; dacă în anumite situații nu veți reuși să vă faceți înțeleși, ați putea pierde oportunități importante datorită unor greșeli stupide.

Este posibil ca cineva să își formeze o părere greșită despre voi dacă, atunci când vă întâlnește prima dată, observă că nu reușiți să vă exprimați coerent și corect.

Evident, nu trebuie să cădeți dintr-o extremă în alta. Am asistat la certuri penibile referitoare la virgule amplasate greșit în mesajele de pe un forum de discuții. Cel care a semnalat erorile s-a supărat că ceilalți nu îi acceptă punctul de vedere și a renunțat să mai folosească acel forum.

În schimb, atunci când ***redactați*** un document important, trebuie să fiți atenți. Eventualele erori strecurate într-un astfel de document pot avea consecințe mai grave decât vă așteptați. Am cunoscut o persoană care nu a fost angajată de o firmă deoarece *CV*-ul trimis conținea erori gramaticale. Chiar dacă era bine pregătit din punct de vedere profesional pentru acel post, candidatul nu a mai avut ocazia să dovedească acest lucru.

Vă sugerăm să nu vă supărați atunci când vi se spune că ați comis astfel de greșeli. Trebuie să fiți capabili să acceptați că ați greșit și să încercați să nu mai comiteți aceleași erori în viitor.

Probabil că puțini dintre voi vor avea de suferit datorită erorilor gramaticale dar, cu siguranță, câțiva veți avea neplăceri exact atunci când nu vă așteptați. Să nu neglijați modul în care vă exprimați considerând că există lucruri mult mai importante decât talentul de a vorbi și a scrie corect.

Concluzii

Am început cu semnalarea unor deficiențe privind exprimarea care, din păcate, apar și în articolele trimise spre publicare și am ajuns să trecem în revistă o mulțime de erori care apar în cele mai diverse situații chiar și în materiale publicate.

Sperăm că, *cititori* și *scriitori*, cu toții ați înțeles că nu am avut intenția să supărăm pe nimeni, în fond este vorba de interesele noastre comune. Dacă un articol reușit este apreciat de mulți cititori, este bucuria noastră, a tuturora. Dar pentru aceasta este nevoie de un grad sporit de atenție în timpul formulării propozițiilor, pentru a vă asigura că ele sunt clare și exprimă exact ceea ce ați vrut să spuneți.

Sperăm că unii dintre voi vor dori să publice articole în *GInfo*. Știm că nu este foarte ușor, mai ales pentru un începător. Citiți încă o dată acest editorial, țineți cont de cele câteva sfaturi și trimiteți articolul vostru cu încredere redacției ***GInfo***. Vă vom sprijini și ajuta și în viitor.

Mihai Scorțaru

Athlon XP

La data de 9 octombrie 2001 compania Advanced Micro Devices a lansat un nou procesor. O dată cu acesta a apărut o nouă arhitectură în industria microprocesoarelor - QuantiSpeed - de unde și terminația XP (eXtreme Performance).

Scurt istoric

Până acum câțiva ani, un indicator eficient al performanțelor procesoarelor era reprezentat de *megahertzi*, deoarece majoritatea utilizatorilor foloseau calculatoare bazate pe aceeași arhitectură.

După ce au fost concepute microprocesoare bazate pe arhitecturi noi, performanța acestora nu mai depindea exclusiv de frecvența lor. În momentul de față performanța se calculează ca fiind produsul dintre frecvență și numărul de instrucțiuni efectuate într-un ciclu, cel puțin așa susține *AMD*.

Lansare

Cel mai performant procesor de pâna acum, destinat calculatoarelor de birou, a fost lansat oficial pe data de 9 octombrie 2001. Tot la această dată compania *AMD* a anunțat un plan care urmează să dezvolte o modalitate sigură, de măsurare a performanțelor calculatoarelor, în care utilizatorii să aibă mai multă încredere.

Inițiativa lor de calcul real al performanțelor *TPI* (*True Performance Initiative*) este menită să-i ajute pe utilizatorii de calculatoare să înțeleagă beneficiile aduse de performanțele lor, și să măsoare performanțele procesoarelor în cazul aplicațiilor standard.

Perfomanțe

Procesorul *AMD Athlon XP*, cunoscut înaintea lansării sale sub numele de cod *Palomino*, folosește un nou model de arhitectură patentată de compania *AMD - QuantiSpeed*. Aceasta oferă o performanță cu aproximativ 25% mai mare decât cea a procesoarelor cu care concurează.

Arhitectura *QuantiSpeed* permite procesorului să prelucreze un volum mai mare de date în timpul unui ciclu de ceas.

Ca o consecință a acestui fapt, chiar dacă alte procesoare au frecvențe mai ridicate, nu prelucrează același volum de date la fel de rapid ca și un *AMD Athlon XP*. Deci, noua arhitectură îi asigură procesorului un "salt în timp" când se lucrează cu aplicații folosite de utilizatorii obișnuiți.

Performanțele noului procesor *AMD Athlon XP* sunt semnificativ mai mari, comparativ cu cele ale predecesorului său *AMD Athlon*.

Procesorul are implementată, de asemenea, și tehnologia *3DNow!* care include 52 de instrucțiuni noi, îmbunătățind astfel performanțele aplicațiilor media, cum ar fi editarea audio și video. Acest procesor oferă performanțe ridicate și în materie de jocuri, oferind *gamer*-ilor un suport deosebit.

Pe lânga arhitectura *QuantiSpeed* și suportul *3DNow!* îmbunătățit, procesorul are încorporată o memorie *cache* de 384 KB care lucrează la aceeași frecvență cu procesorul.

Procesoarele *AMD Athlon XP* sunt produse, la fel ca și procesoarele *Athlon* și *Duron*, folosind o tehnologie de 0,18 microni, bazată pe cupru.

Compania *AMD* va identifica diferitele modele ale noilor procesoare *Athlon XP* prin numere de model (în loc de frecvența acestora exprimată în MHz) cum sunt 1800+, 1700+, 1600+ și 1500+. Procesorul cu modelul numărul 1800+ rulează la o frecvență de 1,53 GHz, numărul 1800 exprimând performanțele procesorului *Athlon XP* (și nu frecvența acestuia) relativ la celelalte procesoare din familia *Athlon*. Celelalte procesoare *Athlon XP* funcționează la frecvențe de 1,47 GHz, 1,4 GHz, respectiv 1,33 GHz.

Pe data de 5 noiembrie a fost lansat un nou procesor din această familie: *Athlon XP* 1900+ care funcționează la frecvența de 1,6 GHz.

Concluzii

Chiar dacă procesoarele produse de compania *AMD* au atins bariera de 1 GHz înaintea celor produse de *Intel*, tehnologia nu i-a permis companiei *AMD* să ajungă prima și la 2 GHz.

Creșterea frecvenței procesoarelor se putea face numai cu riscul de a executa mai puține instrucțiuni pe ciclu, deci acest lucru nu ar fi îmbunătățit semnificativ performanțele procesoarelor ci ar fi condus la creșterea prețurilor de producție și, în consecință a celor de vânzare. Cu toate acestea procesorul *Athlon XP 1900+*, îl surclasează pe rivalul său de la *Intel*, procesorul *Pentium 4* la 1900 MHz.

Windows XP

Compania Microsoft a lansat pe data de 25 octombrie un nou sistem de operare destinat atât utilizatorilor obișnuiți cât și companiilor. Au fost aduse unele modificări revoluționare, motiv pentru care Windows XP a devenit produsul acestei luni.

Scurt istoric

Trecerea de la *Windows Me/2000* la *Windows XP* nu este la fel de spectaculoasă cum a fost trecerea de la *Windows 3.11* la *Windows '95*. (*Windows 3.11* era doar o platformă de operare pentru sistemul de operare *MS-Dos*.) *Windows '95* este un adevărat sistem de operare cu interfață grafică menită să ușureze folosirea calculatorului de toate categoriile de utilizatori. Și chiar fără schimbări spectaculoase, noul sistem de operare de la *Microsoft*, lansat pe data de 25 octombrie aduce o serie de îmbunătățiri și facilități, atât în ceea ce privește interfața grafică cât și managementul mai bun al resurselor.

Noutăți

Noul sistem de operare este disponibil în două variante: *Windows XP Home Edition* și *Windows XP Professional*. Prima variantă se adresează utilizatorilor obișnuiți, iar cea de-a doua este destinată companiilor.

Pe un calculator având configurația recomandată de *Microsoft*, dotat cu 128 MB Ram, *Windows XP* se încarcă în memorie mai repede decât predecesorii săi *Windows 98 SE* și *Windows Me/2000*, și folosește optim resursele sistemului. Performanțele acestuia nu scad prea mult nici dacă se utilizează mai multe aplicații în același timp.

Flexibilitate

Windows XP are implementat un nou nivel de stabilitate. Dacă în sistemele de operare precedente se întâmpla ca un program să se blocheze, acest lucru ducea des la blocarea sistemului de operare, însă acum sistemul de operare continuă să funcționeze indiferent de cauzele care au dus la blocarea programului respectiv.

Folosirea calculatorului de către alți utilizatori se face acum mult mai simplu. Cel care folosește sistemul nu mai este nevoit să închidă aplicațiile deschise de utilizatorul precedent, deoarece sistemul de gestionare al celor care folosesc același calculator este îmbunătățit.

O altă facilitate a sistemului este faptul că permite asistență *on-line*, care poate fi făcută de pe un sistem pe care rulează un prieten sau de o firmă competentă care vă poate ajuta în caz de nevoie, în timp real.

Varianta *Windows XP Professional* permite crearea unei sesiuni virtuale care vă permite să utilizați calculatorul de pe orice alt sistem pe care rulează orice sistem de operare *Windows* cu versiune ulterioară *Windows 3.11*.

Pentru a ușura comunicarea între utilizatorii de calculatoare în timp real, *Windows XP* are integrat *Windows Messenger*, un program cu ajutorul căruia pot fi trimise în rețea mesaje text, mesaje vocale sau în diferite formate video care au o calitate foarte bună. În același timp puteți vedea dacă un utilizator este conectat în sistem.

Interfața grafică utilizator este acum mai colorată și mult mai prietenoasă. *Windows XP* vă ajută să accesați mai rapid aplicațiile folosite mai des, iar suportul multimedia este altul.

Securitate

Revenirea la o stare stabilă a sistemului de operare este asigurată de un utilitar care face copii ale stării sistemului la momente de timp stabilite de anumiți parametri. Acest program este *System Restore*. Acesta vă permite să creați, dacă doriți și manual o copie a stării curente a sistemului. Dacă sistemul nu mai funcționează în parametri normali, puteți să reveniți la o stare precedentă a sistemului.

Windows XP Professional introduce un sistem de criptare a fișierelor care oferă un grad ridicat de securitate, prin urmare o protecție mai bună împotriva *hacker*-ilor, prin codificarea transparentă a fișierelor cu o cheie generată aleator.

Pentru navigarea pe Internet, *Windows XP* folosește ca browser noul *Internet Explorer 6.0* care permite accesul la paginile *Web* cu un grad avansat de securitate.

Informații suplimentare

Pentru detalii referitoare la noul produs al companiei *Microsoft*, puteți vizita *site*-ul oficial dedicat sistemului de operare *Windows XP* la adresa:

http://www.microsoft.com/windowsxp.

Civilization III

Seria jocurilor de strategie Civilization continuă. Spiritul creativ al lui Sid Meyer, l-a făcut să creeze Civilization III, un joc de strategie bazat pe ture, în care jucătorii se întrec cu cei mai mari conducători ai lumii în încercarea de a construi cel mai puternic imperiu.

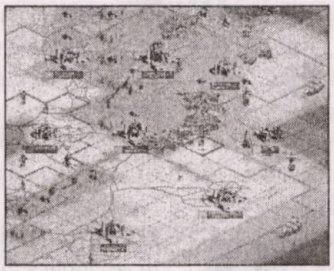
Lansare

Jocul *Civilization III*, succesorul unuia dintre cele mai apreciate jocuri ale deceniului trecut, a fost lansat pe data de 30 octombrie 2001.

Civilization III a fost realizat de compania *Firaxis* și este distribuit de *Infogrames*.

Modul de joc

Civilization III este un joc de strategie, care îmbină elementele din versiunile precedente cu altele noi, majoritatea preluate din *Sid Meyer's Alpha Centauri*, ceea ce a dus la obținerea unui joc complex, dar, în același timp, ușor de utilizat.



La începutul unui joc nou veți fi puși în situația de a alege civilizația al cărei conducător doriți să fiți. Există mai multe civilizații și niveluri de dificultate diferite. Durata de joc crește proporțional cu dificultatea. Dacă ați jucat cel puțin o dată *Civilization II*, va fi imposibil să nu observați câteva dintre schimbările introduse de noua versiune.

Jocul poate fi controlat cu ajutorul *mouse*-ului și al tastaturii. Interfața este similară celei din *Civilization II*. Noua versiune este mult mai flexibilă în ceea ce privește evoluția civilizațiilor în timp.

Dacă în *Civilization II* erați obișnuiți ca toate civilizațiile să aibă aceleași posibilități de dezvoltare, acum fiecare civilizație are la dispoziție unități inaccesibile celorlalte. Cultura și mai târziu tehnologia, au un rol foarte important.

Strategia de joc este diferită de la o civilizație la alta.

În timpul jocului vă veți bucura de un tip nou de hartă, care este "vie" deoarece pe parcurs apar elemente noi.

În momentul construirii unui oraș nou, trebuie să fiți atenți la împrejurimile pe care le va avea, adică resurse și căi de comunicare cu celelalte orașe, deoarece acestea vor influența dezvoltarea lui. Un oraș cu o cultură vastă se va mări și va "înghiți" orașele mai puțin "dotate" din împrejurimi.

Începând cu această variantă a jocului apare conceptul nou de ***graniță interstatală***, care influențează dezvoltarea civilizației.

Arborele tehnologic este mult mai complex, ca de altfel și comerțul, care în perioada secolului al *XX*-lea nu se mai realizează la fel de primitiv ca în *Civilization II*.

La fel ca în *Civilization II* există mai multe modalități de victorie.

În momentul în care ați atins un anumit nivel de cultură sau un anumit nivel tehnologic, aveți la dispoziție posibilitatea de a începe construcția uneia dintre ***cele șapte minuni ale lumii*** (dacă aceasta nu a fost deja construită) sau de a realiza cercetări științifice de amploare (de exemplu, descoperirea inteligenței extraterestre).

Creatorii jocului s-au concentrat mult asupra *AI*-ul (*Artificial Intelligence*). Cu toate că algoritmul ales nu este optim (în joc se observă anumite acțiuni ciudate la prima vedere) adversarii controlați de calculator reușesc să îmbine destul de bine toate atuurile pe care le au la dispoziție, deci vor fi competitori redutabili.

Sistemul necesar

Civilization III nu necesită un calculator foarte performant. Configurația recomandată de producător este un procesor *Pentium II* cu frecvența de 300 MHz sau unul echivalent, 32 MB de memorie RAM, placă video compatibilă *DirectX* 8.0*a* care să accepte rezoluția de 1024 × 768 cu adâncimea de culoare de 16 biți, CD-ROM cu viteza 4× sau mai bun, 450 MB spațiu liber pe disc și, opțional, placă de sunet compatibilă *DirectX* 8.0*a*.

Recomandări

Civilization III este un joc de lungă durată și conține o grafică foarte bogată în detalii, fapt care poate avea efecte negative asupra vederii. În consecință după fiecare oră de joc ar fi bine să faceți o pauză de cel puțin 15 minute.

REFERATE

O mare problemă a elevilor de liceu o constituie referatele cerute de profesori la cele mai diverse și mai "interesante" materii. Mulți nu găsesc materialele necesare realizării unui referat dintr-un anumit domeniu, iar altora nu le convine să facă efort pentru a fi "originali". Pentru a-i ajuta pe cei aflați în dificultate vă prezentăm un site creat pentru a le face viața mai ușoară. Așadar, site-ul acestei luni este www.referate.ro.

Informații generale

Referatele au devenit o problemă pentru mulți elevi de liceu. Unii dintre ei se pare că au găsit o soluție, spre deosebire de alții care încă se mai întreabă dacă nu cumva există undeva pe *Internet*, mai precis pe pagini *Web* românești, referate complete.

Răspunsul la această întrebare este *da*. Există pagini pe *Internet* și chiar *site*-uri de unde se pot descărca referate din diferite domenii. Unul dintre aceste *site*-uri este **www.referate.ro.**

Site-ul oferă, tuturor vizitatorilor, posibilitatea de a publica materialele create de ei sau de a descărca referatele dorite. Pentru a trimite o lucrare se alege *link*-ul de trimitere a referatelor de pe pagina principală, și se selectează fișierele de pe disc ce urmează a fi publicate.

Oricine are nevoie de un referat poate accesa acest *site* pentru a descărca informațiile de care are nevoie în vederea realizării temei.

Aici puteți găsi referate din domeniile: limba și literatura română, engleză, franceză, istorie, geografie, biologie, matematică, astronomie, informatică, economie, fizică, chimie, filosofie, psihologie etc.

Cu toate că materialele oferite de cei care colaborează la îmbunătățirea *site*-ului nu acoperă întreaga programă școlară, referatele disponibile *online* sunt unele dintre cele mai interesante și mai bine realizate.

Pe lângă referate, pe *site* sunt disponibile programele pentru examenele de capacitate și bacalaureat din anul 2002. Tot aici veți găsi modele de subiecte pentru BAC 2002.

Concluzii

Dacă aveți referate mai bune sau din domenii care nu au fost atinse în cadrul acestui *site* le puteți publica pentru a-i ajuta pe cei care nu au posibilitatea să creeze astfel de lucrări. O altă variantă ar fi să ni le trimiteți la redacție și vor fi publicate în *GInfo*. Dar vă reamintim, în *GInfo* se publică doar articole care au subiecte referitoare la informatică.

www.ginfo.ro

Începând cu data de 1 noiembrie 2001 este disponibil noul site al revistei noastre.

Acest nou site al Gazetei de Informatică va încerca, pe cât posibil, să completeze informațiile pe care le găsiți în revistă. Vor exista articole care vor fi publicate doar pe site, articole din revistă care vor face referire la resurse suplimentare disponibile pe site (de exemplu, codurile sursă ale algoritmilor prezentați) etc.

Site-ul conține numeroase secțiuni dintre care amintim: concursul Bursele Agora, probleme propuse și rezolvate, știri TIC de ultimă oră, informații despre hardware, software și jocuri, bacalaureat, concursuri online organizate de GInfo, olimpiade etc.

De asemenea, pe site veți găsi cuprinsul numărului curent, precum și articolele publicate în numerele anterioare.

Zalaegerszeg, Ungaria...

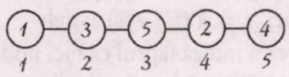
PROBLEME propuse pentru REZOLVARE

Vă prezentăm enunțurile celor șase probleme propuse spre rezolvare concurenților de la Olimpiada de Informatică a Europei Centrale, desfășurată în perioada 10-17 august 2001 în orașul Zalaegerszeg din Ungaria.

P070101: Lanț de atomi

Oamenii de știință studiază un tip special de molecule. Se știe că o moleculă este formată din N atomi diferiți, numerotați de la 1 la N și că molecula are o structură liniară. Oamenii de știință au în dotare un dispozitiv pentru măsurarea distanțelor. La o măsurătoare, instrumentul poate determina distanța dintre doi atomi ai moleculei. Distanța determinată de instrument indică valoarea absolută a diferenței dintre pozițiile în lanț ale celor doi atomi. Dacă nu se accesează nici un atom de mai mult de trei ori, molecula nu suferă nici o deteriorare. Dacă unul sau mai mulți atomi sunt accesați de patru ori, molecula va suferi o mică deteriorare. Dacă unul sau mai mulți atomi sunt accesați de cinci sau mai multe ori, molecula este distrusă.

Sarcina voastră este de a scrie un program care descoperă structura completă a moleculei, determinând ordinea atomilor din moleculă.



Biblioteca

Pentru a utiliza dispozitivul de măsurare a distanțelor aveți la dispoziție o bibliotecă cu numele `meter` care poate efectua trei operații (funcții):

- `Size` - trebuie apelată o singură dată la început, nu are argumente și returnează valoarea N. `Size` trebuie apelată înaintea oricărui apel al funcției `Span`.
- `Span` - este apelată cu doi parametri (etichetele a doi atomi) și returnează distanța dintre cei doi atomi.
- `Answer` - trebuie apelată la sfârșit și este folosită pentru a trimite soluția voastră. `Answer` are doi parametri i și x (numere întregi) și specifică faptul că eticheta atomului de pe poziția i a moleculei este x. Pentru fiecare i ($1 \le i \le N$), `Answer` trebuie apelată exact o dată în ordinea *crescătoare* a valorii i. Există întotdeauna două soluții simetrice; puteți determina doar una dintre ele.

Dialogul dintre programul vostru și bibliotecă va fi descris automat în fișierul text **`chain.out`**. Acest fișier este generat în urma apelurilor subprogramelor din bibliotecă și va conține, de asemenea, răspunsul vostru, numărul maxim de accesări ale atomilor și eventualele mesaje de eroare.

Atenție!!!

Nu programul vostru trebuie să creeze acest fișier; el este generat de bibliotecă.

Instrucțiuni pentru programatorii în Pascal

- În codul sursă folosiți instrucțiunea de includere `uses meter;`.

Instrucțiuni pentru programatorii în C/C++

- În codul sursă folosiți instrucțiunea `#include "meter.h";`.
- Creați în directorul problemei un fișier proiect cu numele **`chain.gpr`**, adăugați fișierele **`chain.c`** (**`chain.cpp`**) și **`meter.o`** în acest proiect și apoi *compilați* și/sau *executați* programul.

Experimentare

Puteți realiza experimente cu biblioteca, creând un fișier text cu numele **`chain.in`**. Fișierul va trebui să conțină două linii. Prima dintre acestea va conține numărul întreg N care reprezintă numărul atomilor din moleculă. A doua linie va conține un set de date de test: o secvență de etichete care constă în N numere întregi distincte cu valori cuprinse între 1 și N.

Exemplu pentru experimentare

`chain.in`

```
5
1 3 5 2 4
```

```
chain.out
Size=5
Span(1,2)=3
Span(1,3)=1
Span(2,3)=2
Span(4,5)=2
Span(1,4)=4
Span(3,5)=1
Your answer:
1 3 5 2 4
Max. Access/Atom:
3
```

Restricții

- Pentru numărul de atomi N, avem $5 \leq N \leq 10000$.
- Accesarea de către funcția `Span` a unui atom de mai mult de patru ori va întrerupe execuția programului vostru.
- Programul nu va trebui să citească date din nici un fișier sau să scrie date într-un anumit fișier.
- Pentru etichetele atomilor (x) și pozițiile atomilor (i), avem $1 \leq i, x \leq N$.
- Denumirile fișierelor bibliotecii pentru limbajul *FreePascal* sunt: **`meter.ppw`** și **`meter.ow`**.
- Declarațiile subprogramelor *Pascal* sunt:

```
function Size:Integer;
function Span(x,y:Integer):Integer;
procedure Answer(i,x:Integer);
```

- Denumirile fișierelor bibliotecii pentru limbajul *C/C++* sunt **`meter.h`** și **`meter.o`**.
- Declarațiile funcțiilor *C/C++* sunt:

```
int Size(void);
int Span(int x,int y);
void Answer(int i,int x);
```

Punctare

Dacă răspunsul vostru este corect și nici un atom nu a fost accesat de mai mult de trei ori, veți primi întregul punctaj. Dacă răspunsul vostru este corect, dar unul sau mai mulți atomi au fost accesați de patru ori, veți obține 50% din punctaj. Dacă răspunsul vostru este incorect sau un anumit atom a fost accesat de cinci ori, nu veți primi nici un punct.

Precizări

- Timpul de execuție este de 1 secundă/test.
- Limita de memorie este de 32 MB.
- Limita pentru dimensiunea sursei este de 1 MB.
- Opțiunile de compilare pentru *FreePascal* sunt:

```
-So -O2 -XS.
```

- Opțiunile de compilare pentru *C/C++* sunt:

```
-O2 -static meter.o.
```

P070102: Circuit imprimat

Un *circuit imprimat* este o placă formată din *noduri* și *segmente corespunzătoare firelor* care leagă perechi de noduri.

Considerăm un circuit imprimat special în care nodurile sunt aranjate pe un caroiaj dreptunghiular și firele leagă (pe verticală sau orizontală) doar noduri adiacente. Un circuit imprimat este *conex* dacă oricare două noduri distincte sunt legate prin cel puțin o secvență de fire. Se dă un circuit imprimat în care există deja câteva fire care leagă câteva noduri adiacente. Trebuie adăugate fire noi astfel încât circuitul imprimat să devină conex. Costul unui fir este `1` dacă este vertical și `2` dacă este orizontal.

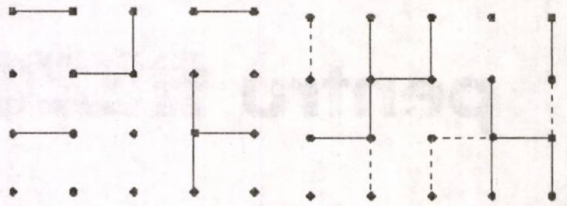


Figura 1

Figura 2

Trebuie să scrieți un program care determină un cost minim pentru obținerea unui circuit conex. Programul vostru va trebui să rezolve următoarele trei subprobleme:

A. Determinarea numărului de fire noi adăugate pentru obținerea unui circuit conex de cost minim.
B. Calcularea valorii costului minim.
C. Determinarea unei liste de fire pentru obținerea circuitului conex de cost minim.

Intrare

Prima linie a fișierului **`circuit.in`** conține două numere întregi, N și M. N ($1 \leq N \leq 100$) este numărul de linii și M ($1 \leq M \leq 100$) este numărul de coloane ale caroiajului. Nodurile circuitului sunt identificate prin coordonatele lor; nodul din colțul stânga-sus are coordonatele (1, 1), iar nodul din colțul dreapta-jos are coordonatele (N, M). Fiecare dintre următoarele N linii conține M numere întregi. Numărul din coloana j a liniei i descrie firele dintre perechile de noduri (i, j) și $(i + 1, j)$, și, de asemenea, dintre perechile de noduri (i, j) și $(i, j + 1)$ astfel:

- valoarea `0` indică faptul că nici nodurile din perechea (i, j) cu $(i + 1, j)$ și nici nodurile din perechea (i, j) cu $(i, j + 1)$ nu sunt legate prin nici un fir.
- valoarea `1` indică faptul că doar nodurile din perechea (i, j) cu $(i + 1, j)$ sunt legate printr-un fir.
- valoarea `2` indică faptul că doar nodurile din perchea (i, j) cu $(i, j + 1)$ sunt legate printr-un fir.
- valoarea `3` indică faptul că atât nodurile din perechea (i, j) cu $(i + 1, j)$, cât și nodurile din perechea (i, j) cu $(i, j + 1)$ sunt legate prin câte un fir.

Există anumite poziții în care nu toate cele patru valori posibile sunt valide (de exemplu, în poziția (N, M) doar valoarea `0` este validă).

Ieșire

Prima linie a fișierului **`circuit.out`** trebuie să conțină două numere întregi, K și V. K este numărul de fire noi care trebuie adăugate pentru obținerea unui circuit imprimat

conex de cost minim. V este valoarea costului minim. Următoarele K linii reprezintă lista firelor care trebuie adăugate pentru obținerea circuitului imprimat conex de cost minim. Fiecare linie va conține trei numere întregi (i, j și d) care vor descrie exact un fir nou. (i, j) sunt coordonatele unui nod, iar d poate avea fie valoarea `1`, fie valoarea `2`.

Valoarea `1` indică faptul că noul fir leagă nodurile (i, j) și ($i + 1$, j), iar valoarea `2` indică faptul că noul fir leagă nodurile (i, j) și (i, $j + 1$).

Exemplu

Fișierul de intrare din exemplu corespunde circuitului din Figura 1. Fișierul de ieșire reprezintă o modalitate posibilă de a obține circuitul imprimat conex de cost minim și este prezentat în Figura 2.

circuit.in

```
4 5
2 1 1 2 1
0 3 0 1 0
3 0 0 3 1
0 2 0 2 0
```

circuit.out

```
5 6
1 1 1
3 2 1
3 3 1
3 3 2
2 5 1
```

Punctare

Dacă răspunsul vostru este corect pentru subpunctul A. veți obține 1 punct. Veți obține încă două puncte dacă răspunsul este corect *și* pentru subpunctul B. Dacă răspunsul este corect pentru toate cele trei subpuncte, veți obține 5 puncte.

Pentru a obține punctele corespunzătoare subpunctelor A. și B. nu este necesar ca fișierul de ieșire să conțină date pentru subpunctul C.

Precizări

- Timpul de execuție este de 1 secundă/test.
- Limita de memorie este de 32 MB.
- Limita pentru dimensiunea sursei este de 1 MB.
- Opțiunile de compilare pentru *FreePascal* sunt:
  `-So -O2 -XS`.
- Opțiunile de compilare pentru *C/C++* sunt:
  `-O2 -static`.

P070103: Călătorie în circuit

Echipa ta decide ca după terminarea competiției să facă o excursie în circuit prin țara gazdă. Doriți să vă deplasați într-un oraș destinație și să vă întoarceți în orașul de plecare. Singura cerință pentru echipa voastră este ca traseul de plecare și traseul de întoarcere să conțină cel mai mic număr posibil de străzi comune. (Un traseu nu poate să conțină aceeași stradă de două sau mai multe ori.)

Trebuie să scrieți un program care determină două trasee între orașul de plecare și orașul destinație, astfel încât numărul de străzi comune ale celor două trasee să fie cât mai mic posibil.

Intrare

Prima linie din fișierul **trip.in** conține două numere întregi, S și D ($S \neq D$) care reprezintă eticheta orașului de plecare și respectiv a orașului destinație. A doua linie conține două numere întregi, N și M, unde N ($3 \leq N \leq 1000$) este numărul de orașe și M ($2 \leq M \leq 100000$) este numărul de străzi care leagă orașele. Orașele sunt etichetate cu numere naturale cuprinse între 1 și N. Fiecare dintre următoarele M linii din fișier conține două numere întregi, P și Q ($1 \leq P, Q \leq N, P \neq Q$), având semnificația că există o stradă cu sens dublu de circulație între orașele P și Q. Există cel mult o stradă între oricare două orașe.

Ieșire

Prima linie a fișierului **trip.out** trebuie să conțină un număr întreg reprezentând cel mai mic număr posibil de străzi comune ale celor două trasee. A doua linie trebuie să conțină traseul de plecare sub forma unei secvențe de etichete ale orașelor, inclusiv orașul de plecare și cel destinație. A treia linie trebuie să conțină traseul de întoarcere, sub forma unei secvențe de etichete ale orașelor, inclusiv orașul destinație și cel de plecare. Dacă există mai multe perechi de trasee având același număr minim de străzi comune, programul vostru poate să furnizeze la ieșire oricare dintre acestea. Dacă nu există nici un traseu de la orașul de plecare la orașul destinație, atunci prima și singura linie a fișierului de ieșire trebuie să conțină doar valoarea `-1`.

Exemplu

trip.in

```
1 6
7 8
2 1
1 3
2 3
4 2
4 5
5 6
7 5
6 7
```

trip.out

```
2
1 3 2 4 5 7 6
6 5 4 2 1
```

Punctare

Dacă prima linie a fișierului de ieșire conține răspunsul corect, veți obține 2 puncte. Dacă prima linie conține soluția

corectă și a doua și a treia linie conțin trasee corecte, veți obține încă 3 puncte.

Precizări

- Timpul de execuție este de 1 secundă/test.
- Limita de memorie este de 32 MB.
- Limita pentru dimensiunea sursei este de 1 MB.
- Opțiunile de compilare pentru *FreePascal* sunt:
  `-So -O2 -Xs`.
- Opțiunile de compilare pentru *C/C++* sunt:
  `-O2 -static`.

P070104: Hartă de biți

Pozele alb-negru sunt deseori stocate sub forma unor hărți de biți. O *hartă de biți* este un caroiaj dreptunghiular de pixeli.

O *linie poligonală* între pixelii P și Q este o secvență de pixeli negri $P = P_1, P_2, ..., P_k = Q$, unde P_i și P_{i+1} ($i = 1, ..., k - 1$) sunt pixeli adiacenți (pe orizontală sau verticală, dar nu pe diagonală). O linie poligonală $P_1, P_2, ..., P_k$ este *închisă* dacă $P_1 = P_k$, și $P_i \neq P_j$ ($i = 1, ..., k - 1$, $j = 2, ..., k$) pentru $i < j$ cu excepția cazului în care $i = 1$ și $j = k$ (adică linia poligonală nu conține același pixel de două ori).

O *mulțime de pixeli negri* S este *conexă* dacă pentru fiecare pereche de pixeli (P, Q) din S există cel puțin o linie poligonală L între P și Q astfel încât toți pixelii de pe linia poligonală L fac parte din mulțimea S.

O *componentă* a unei hărți de biți este o mulțime conexă maximală de biți negri. O componetă poate conține în interior *goluri*. Un gol este format din pixeli albi care se află în interiorul unei linii poligonale închise. O *componentă compactă* nu conține în interior ***nici un gol***.

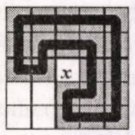


Figura 1

Atenție!!!

Pixelul alb marcat cu x aflat în mijlocul Figurii 1 nu se află în interiorul liniei poligonale marcate.

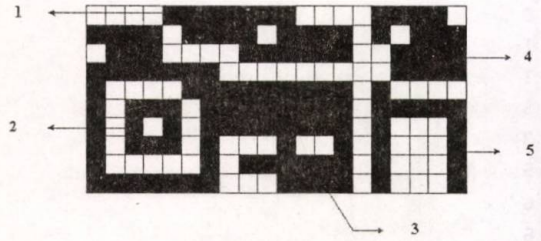


Figura 2

În Figura 2 se află o hartă de biți cu cinci componente, dintre care două sunt compacte.

Scrieți un program care determină numărul total de componente și numărul de componente compacte ale unei hărți de biți.

Codificare

Hărțile de biți investigate sunt codificate (comprimate, arhivate), folosind metoda descrisă în continuare.

Fiecare linie este codificată printr-o secvență de numere întregi $W_1, B_1, ..., W_k, B_k$, unde W_i este numărul de pixeli albi consecutivi, iar B_i este numărul de pixeli negri consecutivi. De exemplu, codificarea primei linii a hărții de biți din Figura 2 este secvența `4 7 4 4 1 0`.

Componentele 4 și 5 sunt compacte, în timp ce componentele 1, 2 și 3 nu sunt compacte.

Intrare

Prima linie a fisierului `bitmap.in` conține două numere întregi pozitive, N și M. N ($2 \leq N \leq 10000$) reprezintă numărul de linii, iar M ($2 \leq M \leq 1000$) reprezintă numărul de coloane ale hărții de biți. Următoarele N linii conțin harta de biți codificată așa cum s-a descris în paragraful referitor la *codificare*. Fiecare linie se termină cu valoarea `-1`.

Ieșire

Fișierul `bitmap.out` trebuie să conțină două linii, cu un singur număr întreg pe fiecare linie. Prima linie conține numărul tuturor componentelor, iar a doua linie conține numărul componentelor compacte din harta de biți de la intrare.

Exemplu

`bitmap.in`

```
10 20
4 7 4 4 1 0 -1
0 4 1 4 1 4 1 1 1 3 -1
1 3 4 6 2 4 -1
0 7 9 4 -1
0 1 4 9 1 1 4 0 -1
0 1 1 3 1 8 1 5 -1
0 1 1 1 1 1 1 8 1 1 3 1 -1
0 1 1 3 1 1 3 1 2 1 1 1 3 1 -1
0 1 5 1 1 6 1 1 3 1 -1
0 7 3 4 1 1 3 1 -1
```

`bitmap.out`

```
5
2
```

Punctare

Dacă prima linie a fișierului vostru de ieșire conține numărul corect al tuturor componentelor, veți obține 50% din punctaj. Dacă a doua linie a fișierului vostru de ieșire conține numărul corect al componentelor compacte, veți obține *încă* 50% din punctaj.

Precizări

- Timpul de execuție este de 10 secunde/test.
- Limita de memorie este de 32 MB.
- Limita pentru dimensiunea sursei este de 1 MB.
- Opțiunile de compilare pentru *FreePascal* sunt:
  `-So -O2 -Xs`.
- Opțiunile de compilare pentru *C/C++* sunt:
  `-O2 -static`.

P070105: Șabloane Wildcard

Șabloanele wildcard sunt folosite frecvent pentru a specifica mulțimi de nume. De exemplu, putem specifica toate numele de fișiere care încep cu `h` și se termină cu `bak` prin șablonul `h*bak`.

Un *șablon wildcard* este un string care poate conține caractere `*` considerate *wildcard*-uri. Un string W se potrivește peste un șablon P dacă W poate fi obținut din P înlocuind fiecare caracter `*` din P cu câte un string (eventual unul vid). (Stringuri diferite pot fi folosite pentru apariții diferite ale caracterului `*`.) Pentru o pereche de șabloane P_1 și P_2, Q este un *șablon comun* al șabloanelor P_1 și P_2 dacă orice string care se potrivește peste Q se potrivește atât peste P_1 cât și peste P_2.

O mulțime $Q_1, Q_2, ..., Q_L$ de șabloane comune este numită *completă* dacă orice string care se potrivește atât peste P_1 cât și peste P_2 se potrivește peste cel puțin un element Q_i al mulțimii.

Trebuie să scrieți un program care, pentru o pereche dată de șabloane P_1 și P_2 determină:

A. cel puțin un șablon comun (dar nu și șabloane incorecte); sau

B. o mulțime completă de șabloane care nu conține mai mult de 6666 de elemente; sau

C. o mulțime completă de șabloane cu număr minim de elemente (adică nu există nici o altă mulțime completă de șabloane care să aibă mai puține elemente).

Atenție!!!

Dacă rezolvați subpunctul C. veți obține un *bonus* de 50% la punctaj.

Intrare

Prima și a doua linie a fișierului `pattern.in` conține șabloanele P_1 și P_2.

Șabloanele sunt formate din litere mici cuprinse între `'a'` și `'z'` și caracterul `'*'`. Fiecare șablon conține cel mult 20 de caractere. Numărul de apariții ale caracterului `*` în fiecare șablon este cuprins între 0 și 6.

Ieșire

Prima linie a fișierului `pattern.out` trebuie să conțină un număr întreg K, care reprezintă numărul șabloanelor care formează soluția. Următoarele K linii conțin câte un șablon comun; acestea sunt șabloanele care formează soluția problemei.

Ordinea șabloanelor comune nu are importanță. Atât subpunctul B. cât și subpunctul C. pot fi rezolvate în limita specificată de 6666 de elemente. Dacă nu există nici un șablon comun al șabloanelor P_1 și P_2, atunci prima și singura linie a fișierului va conține valoarea 0.

Exemplu

`pattern.in`

```
*ab*
ba*b
```

`pattern.out` (pentru subpunctul C.)

```
4
ba*ab*b
bab*b
ba*ab
bab
```

Punctare

Dacă soluția voastră rezolvă subpunctul A. veți obține 5 puncte. Dacă soluția voastră rezolvă subpunctul B. veți obține 10 puncte. Dacă soluția voastră rezolvă subpunctul C. veți obține 15 puncte.

Precizări

- Timpul de execuție este de 7 secunde/test.
- Limita de memorie este de 32 MB.
- Limita pentru dimensiunea sursei este de 1 MB.
- Opțiunile de compilare pentru *FreePascal* sunt:
  `-So -O2 -XS`.
- Opțiunile de compilare pentru *C/C++* sunt:
  `-O2 -static`.

P070106: Grup majoritar

Copiii dintr-o școală fac parte din două grupuri disjuncte. Se știe că unul din cele două grupuri, numit grup majoritar conține mai mulți copii decât celălalt. Se urmărește determinarea unui copil care aparține grupului majoritar. Singura operație care se poate efectua este de a întreba dacă doi copii oarecare aparțin sau nu aceluiași grup.

Scrieți un program care determină un copil care aparține grupului majoritar, punând *cât mai puține întrebări.*

Biblioteca

Pentru a putea pune întrebări, aveți la dispoziție o bibliotecă `query` care conține trei subprograme:

- `Size` - trebuie apelat o singură dată, la începutul programului, fără parametri; acesta va returna numărul N, reprezentând numărul de copii. Copiii sunt numerotați de la 1 la N. `Size` trebuie apelat înainte de primul apel al subprogramului `Member`.
- `Member` - are doi parametri, reprezentând numerele de ordine a doi copii; dacă cei doi copii aparțin aceluiași grup, subprogramul returnează `1`, altfel returnează `0`.
- `Answer` - se apelează o singură dată la sfârșitul programului și trebuie folosit pentru a transmite soluția voastră. `Answer` are un singur parametru întreg, reprezentând numărul de ordine al copilului care aparține grupului majoritar.

Se ține evidența dialogului dintre programul vostru și bibliotecă și se salvează în fișierul text `select.out`. Acest fișier conține, de asemenea, răspunsul vostru și precizarea dacă acest răspuns este corect sau nu.

Atenție!!!

Nu există teste predefinite, biblioteca produce răspunsuri "din mers". Dacă încercați să dați un răspuns înainte să fiți

100% siguri de corectitudinea lui, primiți 0 puncte; așa că nu încercați să dați cu banul! Răspunsul vostru este acceptat numai dacă pentru oricare două grupuri disjuncte, care corespund răspunsurilor primite la întrebările puse de program, copilul dat de programul vostru ca fiind în grupul majoritar, se află într-adevăr acolo. Biblioteca forțează programul vostru să pună toate întrebările care sunt necesare pentru a identifica un membru al grupului majoritar.

Instrucțiuni pentru programatorii în Pascal

- folosiți instrucțiunea de includere `uses query;` în codul sursă.

Instrucțiuni pentru programatorii în C/C++

- folosiți instrucțiunea `#include "query.h";` în codul sursă;
- creați în directorul problemei un fișier proiect cu numele `select.gpr`, adăugați fișierele `select.c` (`chain.cpp`) și `select.o` în acest proiect și apoi *compilați* și/sau *executați* programul.

Experimentarea

Puteți face experimente cu biblioteca creând un fișier text `select.in`. Prima și singura linie trebuie să conțină numărul de copii N. N trebuie să fie impar!

Exemplu

`select.in`

```
7
```

`select.out`

```
Size=7
Member(1,2)=0
Member(3,4)=1
Member(5,6)=1
Member(4,6)=0
Your Answer: 7, Correct
Majority Group:
2 5..7
Non-majority Group:
1 3 4
Number of Queries: 4
Full Possible Score: 3
Your Score: 3
```

De exemplu, răspunsul `1` nu este acceptat, deoarece, cum toate răspunsurile dau același rezultat pentru grupurile {2, 5, 6, 7} și {1, 3, 4}, `1` nu e membru în grupul majoritar {2, 5, 6, 7}.

Restricții

- $5 \le N < 30000$, N impar.
- Programul vostru nu trebuie să citească date dintr-un fișier sau să scrie date într-un fișier.
- Copiii sunt identificați prin numerele de ordine i, unde $1 \le i \le N$.
- Denumirile fișierelor bibliotecii pentru limbajul *FreePascal* sunt: `query.ppw` și `query.ow`.
- Declarațiile subprogramelor *Pascal*:

```
function Size:Integer;
function Member(x,y:Integer):Integer;
procedure Answer(x:Integer);
```

- Denumirile fișierelor bibliotecii pentru limbajul *C/C++* sunt: `query.h` și `query.o`
- Declarațiile funcțiilor *C/C++*:

```
int Size(void);
int Member(int x,int y);
void Answer(int x);
```

Punctare

Dacă răspunsul vostru este corect și numărul de întrebări pe care programul le-a pus este K, atunci obțineți $max(0, N - K)$ puncte, unde N este numărul de copii.

Precizări

- Timpul de execuție este de 1 secundă/test.
- Limita de memorie este de 32 MB.
- Limita pentru dimensiunea sursei este de 1 MB.
- Opțiunile de compilare pentru *FreePascal* sunt:
  `-So -O2 -XS`.
- Opțiunile de compilare pentru *C/C++* sunt:
  `-O2 -static query.o`.

România la CEOI 2001

La CEOI 2001, România a obținut două medalii de argint (prin Daniel Dumitran din București și Florin Ghețu din Focșani) și una de bronz (prin Victor Costan).

Vencel Bors din Oradea, a fost foarte aproape de a obține o medalie.

Delegația României l-a avut ca team leader pe dl. lect. univ. Stelian Ciurea și ca deputy leader pe dl. prof. Doru Popescu Anastasiu.

Au mai participat, în calitate de invitați, d-na lect. univ. Clara Ionescu, Mihai Scorțaru și Beatrice Ionescu.

Olimpiada INTERNAȚIONALĂ de INFORMATICĂ

Vă prezentăm soluțiile oficiale ale celor șase probleme pe care au încercat să le rezolve participanții la IOI 2001. Implementările algoritmilor prezentați sunt disponibile pentru download la www.ginfo.ro/revista/11_7/main.html.

P060101: Telefoane mobile

Timo Tossavainen, Jyrki Nummenmaa

Este mai ușor să găsim soluția dacă luăm în considerare doar cazul unidimensional (un vector de dimensiune N) pentru care avem actualizări și interogări pentru un interval. După găsirea unei soluții pentru această variantă, va fi relativ simplu să o generalizăm pentru două (sau chiar mai multe) dimensiuni.

Cazul unidimensional

Dacă valorile vectorului sunt memorate în maniera clasică, atunci determinarea sumei unui interval necesită $O(N)$ operații. Răspunsul la o interogare referitoare la suma valorilor dintr-un interval $[X, Y]$ poate fi determinat dacă sunt cunoscute sumele pentru intervalele $[1, X - 1]$ și $[1, Y]$; el va fi calculat scăzând cele două valori. Sumele corespunzătoare intervalelor de forma $[1, I]$ pot fi stocate într-un tabel și, în acest caz, răspunsul la interogările referitoare la intervale de forma $[X, Y]$ poate fi calculat într-un timp $O(1)$. Totuși, dacă dorim să păstrăm aceste sume, o actualizare ar necesita $O(N)$ operații.

Arborele indexat binar este o structură de date care permite determinarea sumelor și efectuarea actualizărilor în timp $O(\log N)$ și nu necesită spațiu suplimentar de memorie, cantitatea de memorie ocupată fiind egală cu cea pe care ar ocupa-o vectorul clasic. În figura 1 este prezentat un astfel de arbore.

Elementele tabloului sunt indexate cu numere cuprinse între 1 și N, iar elementul de la indicele I conține suma corespunzătoare intervalului $[I - 2^k + 1, I]$, unde k reprezintă numărul zerourilor aflate după ultimul 1 din reprezentarea binară a valorii I. Suma corespunzătoare unui interval poate fi determinată folosind $2 \cdot O(\log N)$ operații.

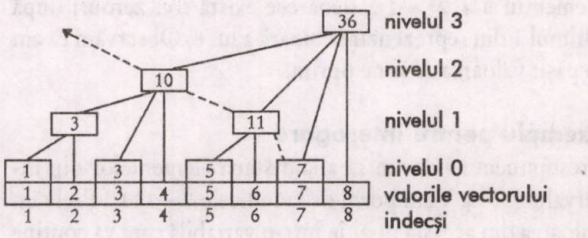


Figura 1. Un arbore indexat binar. Liniile continue indică direcțiile (traseele de la un element la altul) folosite pentru actualizare, iar linia întreruptă indică direcția folosită pentru interogarea referitoare la suma elementelor din intervalul [1, 7] (7 + 11 + 10 = 28). Tabloul inițial nu trebuie memorat; în memorie se află doar liniile care reprezintă niveluri ale arborelui.

În cazul în care numărul de telefoane active dintr-o anumită celulă I crește/scade cu x, va trebui să actualizăm valorile păstrate. O actualizare necesită modificarea tuturor elementelor care păstrează sume ale unor intervale, care conțin și poziția I. Vom porni de la elementul cu indicele I și vom aduna/scădea valoarea x. Pentru a determina următorul element care trebuie actualizat, va trebui să adunăm la I valoarea 2^k unde k reprezintă numărul zerourilor aflate după ultimul 1 din reprezentarea binară a valorii I. Vom continua până în momentul în care obținem o valoare mai mare decât N. Se observă că actualizările necesită $O(\log N)$ operații.

Pentru interogări referitoare la suma elementelor unui interval de forma $[1, I]$ vom porni tot de la elementul cu indicele I. Pentru a determina următorul element va trebui să scădem din I valoarea 2^k, unde k reprezintă numărul zerourilor aflate după ultimul 1 din reprezentarea binară a valorii I. Vom continua până în momentul în care obținem valoarea 0. Se observă că o astfel de interogare necesită $O(\log N)$ operații. Datorită faptului că, pentru a determi-

na suma corespunzătoare unui interval de forma $[X, Y]$, avem nevoie de două astfel de interogări, numărul total de operații va fi $2 \cdot O(\log N)$. Răspunsul la interogare este calculat însumând valorile tuturor elementelor traversate.

Interogările pot fi optimizate pentru intervale mici, dacă se observă că cele două sume care trebuie calculate sunt obținute, folosind anumite elemente comune. Datorită faptului că răspunsul final este dat de diferența celor două sume, se observă că valorile elementelor comune sunt adunate și apoi scăzute. Așadar, căutarea se poate opri în momentul în care se ajunge la un element comun.

Exemplu pentru actualizare

Presupunem că dorim să actualizăm valoarea elementului de pe poziția 3 și avem $N = 8$. Vom porni de la indicele $3 = 11_2$. Vom actualiza valoarea, și apoi vom trece la elementul următor. Nu există nici un 0 după ultimul 1 din reprezentarea binară a lui 3, deci vom trece la poziția $3 + 2^0 = 4 = 100_2$. Actualizăm valoarea și apoi trecem la elementul următor. Există două zerouri după ultimul 1 din reprezentarea binară a lui 4, deci vom trece la poziția $4 + 2^2 = 8 = 1000_2$. După actualizarea valorii vom trece la elementul $8 + 2^3 = 16$, deoarece există trei zerouri după ultimul 1 din reprezentarea binară a lui 8. Observăm că am depășit valoarea N și ne oprim.

Exemplu pentru interogare

Presupunem că dorim să aflăm suma elementelor din intervalul [1, 7]. Vom porni de la indicele $7 = 111_2$. Preluăm valoarea din această poziție într-o variabilă care va conține suma parțială și apoi trecem la pasul următor. Nu există nici un 0 după ultimul 1 din reprezentarea binară a lui 7, deci vom trece la poziția $7 - 2^0 = 6 = 110_2$. Preluăm valoarea și o adunăm la suma parțială, apoi trecem la pasul următor. Există un zero după ultimul 1 din reprezentarea binară a lui 6, deci vom trece la poziția $6 + 2^1 = 4 = 100_2$. Preluăm și această valoarea (adăugând-o la suma parțială) și apoi vom obține valoarea $4 - 2^2 = 0$, deoarece există două zerouri după ultimul 1 din reprezentarea binară a lui 4. Am obținut valoarea 0, deci ne oprim; suma obținută în acest moment corespunde intervalului [1, 7].

Cazul bidimensional

Soluția cazului unidimensional poate fi generalizată pentru oricâte dimensiuni. Pentru rezolvarea acestei probleme sunt necesare două dimensiuni. Arborii sunt construiți în același mod ca și în cazul unidimensional, formându-se un "arbore de arbori". În această structură, elementul de pe poziția $[X, Y]$ va conține suma corespunzătoare dreptunghiului care are colțul stânga sus în poziția $[X - 2^k + 1, Y - 2^l + 1]$, unde k reprezintă numărul zerourilor aflate după ultimul 1 din reprezentarea binară a valorii X, iar unde l reprezintă numărul zerourilor aflate după ultimul 1 din reprezentarea binară a valorii Y.

Folosind această structură vom putea răspunde într-un timp $O((\log N)^2)$ la interogări referitoare la suma elementelor din dreptunghiul care are colțul din stânga-sus în poziția [1, 1] și colțul din dreapta-jos în poziția $[X, Y]$. Pentru a determina suma S a elementelor unui dreptunghi care are colțul din stânga-sus în poziția $[X_1, Y_1]$ și colțul din dreapta-jos în poziția $[X_2, Y_2]$ vom avea nevoie de patru interogări (toate referitoare la dreptunghiuri care au colțul din stânga-sus în poziția [1, 1]):

- dreptunghi cu colțul din dreapta-jos în poziția $[X_2, Y_2]$ (S_1);
- dreptunghi cu colțul din dreapta-jos în poziția $[X_1, Y_2]$ (S_2);
- dreptunghi cu colțul din dreapta-jos în poziția $[X_2, Y_1]$ (S_3);
- dreptunghi cu colțul din dreapta-jos în poziția $[X_1, Y_1]$ (S_4).

Vom avea $S = S_1 - S_2 - S_3 + S_4$.

În cazul bidimensional poate fi realizată aceeași optimizare ca în cazul unidimensional. Astfel, pentru dreptunghiuri mici, căutarea se oprește în momentul în care se găsesc celule comune în momentul calculării unor sume care apar cu semne diferite în formula finală.

Observație

Pentru ca proprietățile folosite pentru actualizare și interogare să fie respectate, indicii folosiți pentru un arbore indexat binar trebuie să înceapă cu valoarea 1. În această problemă, indicii încep cu valoarea 0, deci trebuie efectuată o corecție a acestora (adunarea valorii 1 după citirea lor).

Exemple pentru actualizare

Să presupunem că dorim să actualizăm valoarea elementului de pe poziția (2, 5) și avem $N = 8$. Va trebui să determinăm elementele de pe linia a doua care trebuie actualizate. Pentru o linie folosim același procedeu ca și în cazul unidimensional. Vom actualiza elementele de pe coloanele $5 = 101_2$, $5 + 2^0 = 6 = 110_2$ și $6 + 2^1 = 8 = 1000_2$. Vom trece apoi la linia următoare: $2 + 2^1 = 4 = 100_2$ și actualizăm elementele de pe aceleași coloane. Ultima linie va fi $4 + 2^2 = 8 = 1000_2$ și se vor actualiza elementele de pe aceleași coloane.

În figura 2 sunt prezentate traseele parcurse pentru actualizarea elementului de pe poziția (5, 2).

Vom prezenta și exemplul actualizării elementului de pe poziția (1, 1). Traseele parcurse sunt ilustrate în figura 3.

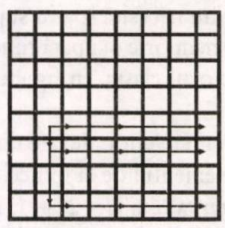


Figura 2. Traseele parcurse pentru actualizarea structurii de date în cazul modificării numărului de telefoane mobile active din celula de coordonate (5, 2)

Exemple pentru interogare

Să presupunem că dorim să aflăm suma elementelor din dreptunghiul care are colțul din stânga-sus în poziția (1, 1)

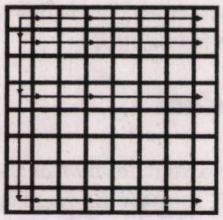


Figura 3. Traseele parcurse pentru actualizarea structurii de date în cazul modificării numărului de telefoane mobile active din celula de coordonate (1, 1)

și colțul din dreapta-jos în poziția (7, 7). Va trebui să determinăm elementele de pe linia a șaptea care trebuie actualizate. Pentru o linie folosim același procedeu ca și în cazul unidimensional. Vom adăuga la suma parțială elementele de pe coloanele $7 = 111_2$, $7 - 2^0 = 6 = 110_2$ și $6 - 2^1 = 4 = 100_2$. Vom trece apoi la linia următoare: $7 - 2^1 = 6 = 110_2$ și vom adăuga la suma parțială elementele de pe aceleași coloane. Ultima linie va fi $6 - 2^1 = 4 = 100^2$ și se vor aduna elementele de pe aceleași coloane.

În figura 4 sunt prezentate traseele parcurse pentru determinarea sumei elementelor din dreptunghiul considerat.

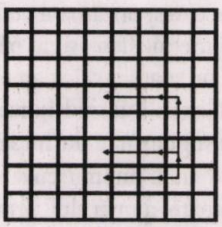


Figura 4. Traseele parcurse pentru determinarea sumei corespunzătoare dreptunghiului care are colțul din stânga-sus în poziția (1, 1) și colțul din dreapta-jos în poziția (7, 7).

Vom prezenta și exemplul determinării sumei elementelor din dreptunghiul care are colțul din stânga-sus în poziția (1, 1) și colțul din dreapta-jos în poziția (3, 6). Traseele parcurse sunt ilustrate în figura 5.

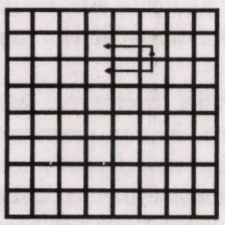


Figura 5. Traseele parcurse pentru determinarea sumei corespunzătoare dreptunghiului care are colțul din stânga-sus în poziția (1, 1) și colțul din dreapta-jos în poziția (3, 6).

Dreptunghiuri corespunzătoare unor celule

În figura 6 sunt reprezentate dreptunghiuri corespunzătoare anumitor celule. Celula este haşurată în negru, iar dreptunghiul corespunzător este haşurat în gri.

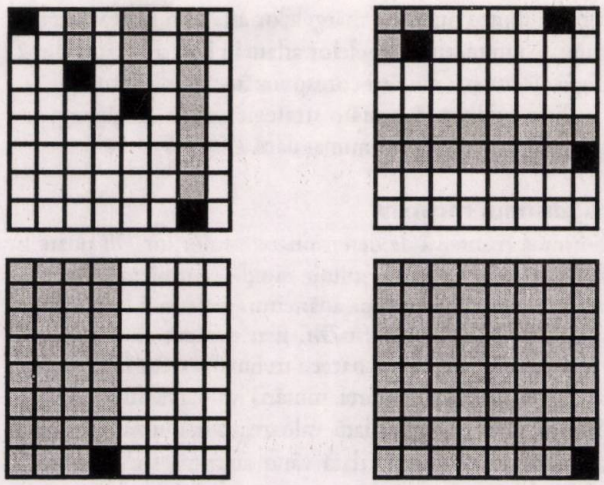


Figura 6. Dreptunghiurile (haşurate în gri) corespunzătoare anumitor celule (haşurate în negru)

P060102: Jocul Ioiwari

Gyula Horvath

Jocul *Ioiwari* este o adaptare a jocurilor existente din familia *Mancala*. Diferite caracteristici (numărul de adâncituri, numărul și distribuția mărgelelor și, în special, regulile jocului) au fost modificați pentru ca următoarele condiții să fie satisfăcute:

- jocul poate fi câștigat de Jucătorul 1;
- în majoritatea situațiilor strategia de câștig nu se găsește ușor;
- există o soluție eficientă care folosește o cantitate rezonabilă de resurse (memorie și timp *CPU*);
- jocul se poate termina la egalitate;
- numărul diferitelor instanțe ale jocului este suficient de mare pentru testarea programelor concurenților.

Considerăm graful orientat ale cărui noduri sunt perechi de forma (w, B) unde w poate avea valoarea 1 sau valoarea 2 și identifică jucătorul aflat la mutare, iar B este o posibilă configurație a tablei (ignorând băncile).

Există un arc de la nodul (u, A) la nodul (v, B) dacă și numai dacă $u + v = 3$ și configurația B poate fi obținută din configurația A, pe baza unei mutări valide. Datorită faptului că, după fiecare mutare, numărul de mărgele din adâncituri scade, graful este aciclic.

Fie $Dif(w, B)$ cea mai mare diferență de scor pe care o poate obține Jucătorul 1 în cazul în care se pornește din configurația B și Jucătorul 2 joacă optim. Ajungem la următoarea formulă:

$Move(B, i)$ reprezintă configurația tablei după efectuarea mutării corespunzătoare adânciturii i (mutarea care începe cu alegerea adânciturii i), iar $D(B, i)$ reprezintă di-

$$Dif(w,B)=\begin{cases}0, \text{ dacă este configurația în care nu mai}\\ \quad\text{există mărgele în nici o adâncitură}\\ \max\left(Dif\left(2,Move(B,i)+D(B,i)\right)\right), \text{pentru}\\ \quad\text{toate mutările valide } i, \text{ dacă } w=1\\ \min\left(Dif\left(1,Move(B,i)+D(B,i)\right)\right), \text{pentru}\\ \quad\text{toate mutările valide } i, \text{ dacă } w=2\end{cases}$$

ferența dintre numărul mărgelelor aflate în banca Jucătorului 1 și numărul mărgelelor aflate în banca Jucătorului 2 după efectuarea mutării corespunzătoare adâncimii *i*.

Este evident că există o strategie sigură de câștig pentru Jucătorul 1, dacă și numai dacă $Dif(1, B) > 0$.

Algoritmul recursiv

Formula recursivă de determinare a valorilor *Dif* poate fi folosită pentru a găsi o soluție rapidă a problemei: Jucătorul 1 va alege întotdeauna adâncitura *i* căreia îi corespunde o valoare maximă pentru *Dif*. Este evident că acest algoritm nu este eficient, deoarece trebuie să determine la fiecare pas (înaintea fiecărei mutări) valoarea funcției *Dif*. Mai mult, este recalculată valoarea corspunzătoare unei configurații de fiecare dată când aceasta este "accesată" (valoarea corespunzătoare apare în formulă).

Algoritmul de memorizare

Putem îmbunătăți eficiența algoritmului recursiv dacă memorăm valorile $Dif(w, B)$ după calcularea lor pentru prima dată și apoi le preluăm de fiecare dată când avem nevoie de ele, fără a mai fi necesară recalcularea lor.

Pentru a memora aceste valori, vom atribui câte un număr unic de identificare pentru fiecare configurație posibilă a tablei. Deoarece, pe parcursul jocului, fiecare adâncitură poate conține cel mult cinci mărgele, fiecare configurație poate fi identificată în mod unic printr-un număr de șapte cifre scris în baza 6.

La început, programul va determina valorile optime pentru subproblemele corespunzătoare determinării valorilor $Dif(w, B)$ și păstrează mutarea optimă într-un tabel. Calculele sunt realizate folosind o tehnică recursivă cu memoizare. În timpul jocului, algoritmul va căuta mutarea optimă în acest tabel.

Cantitatea de memorie necesară este de $3 \cdot MaxN$ bytes, unde *MaxN* este 279935 (cel mai mare număr de șapte cifre scris în baza 6).

Se observă că fiecare configurație este echivalentă cu alte configurații obținute prin rotație. Spațiul necesar poate fi redus la jumătate dacă atribuim fiecărei configurații cel mai mic număr obținut considerând toate configurațiile echivalente obținute prin rotație.

Timpul necesar execuției programului este proporțional cu numărul arcelor grafului (la care se adaugă timpul necesar inițializării tabelelor). O aproximație (prin adaos) a limitei superioare este $7 \cdot MaxN$.

Algoritmi greedy

Poate fi implementat un algoritm care folosește o strategie de tip *greedy*. Algoritmul va "privi în viitor" doar un singur pas (o singură mutare). Jucătorul va alege întotdeauna mutarea care corespunde optimului local. Această strategie este optimă doar pentru câteva instanțe ale jocului. Deși un algoritm, care "privește în viitor" doi pași ar trebui să fie mult mai bun, în practică un astfel de algoritm pierde aproape întotdeauna jocul.

Algoritmul probabilist

O altă posibilitate de rezolvare a problemei o constituie un algoritm care încearcă să ghicească mutarea corectă. Probabilitatea de a câștiga jocul alegând la întâmplare (aleator) mutările efectuate este foarte mică.

P060103: Doi-cinci

Sergey Melnik

Soluția se bazează pe o funcție *numconts* care, pentru o mulțime arbitrară dată, formată din litere care au pozițiile fixate, determină numărul de posibilități de a plasa restul literelor. Se va încerca poziționarea literelor pe marginile unui contur al unui tablou *Young* (același tip de contur care apare și în cazul problemei *Depozit* - vezi pagina 20).

Dacă toate literele au fost poziționate, atunci există o singură posibilitate de continuare: nu se mai realizează nici o altă operație. Deci, pentru o stare completă (toate literele sunt poziționate), *numconts* va inițializa cu `1` valoarea `snum[states - 1]`. În continuare se va completa tabelul folosind funcția *calcstate*.

Funcția *calcstate* returnează numărul posibilităților de a alege poziții pentru literele rămase, dându-se literele poziționate anterior și conturul corespunzător acestei poziționări. Se va încerca poziționarea următoarei litere în toate pozițiile valide și se vor însuma valorile obținute prin autoapeluri recursive corespunzătoare noilor contururi obținute. Rezultatele intermediare sunt păstrate în tabelul *snum* pentru a nu fi necesară recalcularea lor.

Rezolvarea problemei folosind funcția *numconts* este relativ simplă. Să presupunem că dorim să determinăm numărul corespunzător unui anumit cuvânt. Vom păstra o valoare corespunzătoare numărului primului cuvânt (în ordine alfabetică), care poate fi generat folosind literele care sunt fixate în acel moment. Vom fixa literele, una câte una, începând cu cea mai semnificativă. O literă va fi fixată alegând la început valoarea `'A'` și apoi vom alege succesiv valorile următoare până când ajungem la cea dorită. De fiecare dată când alegem o nouă valoare pentru litera curentă, folosim funcția *numconts* pentru a determina numărul cuvintelor peste care "s-a sărit" pe parcursul efectuării operației și valoarea curentă este actualizată corespunzător. După fixarea tuturor literelor am obținut soluția corectă.

Efectuarea operației inverse este asemănătoare. Vom poziționa literele una cîte una, incrementând valorile lor până în momentul în care valoarea curentă depășește valoarea dorită. În momentul apariției depășirii efectuăm "un pas înapoi" (decrementăm valoarea literei), și trecem la litera următoare. La sfârșit, valoarea curentă este egală cu valoarea dorită și am obținut cuvântul corect.

Funcția *calcstate* determină numărul literelor care se află între litera curentă și ultima literă inserată în contur. Folosind acest număr, este posibil să verificăm numărul continuărilor peste care s-a "sărit" alegând continuarea curentă și care au fost "pierdute". Acest număr al continuărilor "pierdute" este scăzut din numărul continuărilor

posibile. Adăugând literele în acest mod (folosind o tehnică recursivă), vom observa că unele litere pot fi așezate într-o singură poziție. Aceste litere sunt fixate; aceste poziții fixe sunt folosite la plasarea literelor.

P060104: Score

Timo Poranen, Jyrki Nummenmaa

Testarea anumitor tipuri de proprietăți ale unui proces cu număr de stări poate fi privită ca un joc *MC*. Câștigarea unui joc *MC* indică dacă proprietatea considerată este respectată pentru un anumit proces. Jocul ***Score*** este o adaptare a jocurilor *MC* de conectivitate a ciclurilor, o categorie specială a jocurilor *MC*. Noțiunea de ***conectivitate a ciclurilor*** indică faptul că toate ciclurile cuprinse într-o componentă tare conexă a grafului jocului au cel puțin un nod comun. Strategia optimă pentru jocul ***Score*** poate fi determinată în timp liniar folosind un algoritm bazat pe căutarea în adâncime.

Din enunțul problemei rezultă că starea inițială a jocului permite întotdeauna alegerea unei săgeți astfel încât, dacă jocul durează destul de mult, jetonul ajunge din nou în poziția inițială. Aceasta înseamnă că ciclurile grafului au cel puțin un nod comun și anume cel corespunzător poziției inițiale.

Strategia de câștig implică alegerea unui drum de la poziția curentă la poziția de pornire în cazul în care după parcurgerea drumului scorul Jucătorului 1 este mai mare decât scorul adversarului. Dacă acest lucru nu este posibil, atunci se alege un drum după parcurgerea căruia scorul adversarului este minim.

Pentru a găsi cea mai bună alegere (cea mai convenabilă săgeată) pentru poziția curentă, se folosește un algoritm care combină căutarea în adâncime cu următoarele reguli recursive de alegere a săgeților:

- dacă există unul sau mai multe drumuri după a căror parcurgere scorul Jucătorului 1 este mai mare decât cel al adversarului, atunci se alege drumul care duce la obținerea celui mai mare scor;
- dacă după parcurgerea oricăruia dintre drumurile care pornesc din poziția curentă, scorul adversarului este mai mare, atunci se alege drumul pentru care scorul adversarului este minim.

Algoritmi greedy

Jucătorul 1 poate folosi o strategie în care "privește în viitor" un singur pas. El va alege întotdeauna o săgeată care va duce la o poziție a cărui posesor este și căreia îi corespunde o valoare cât mai mare.

Dacă nici o săgeată care pleacă din poziția curentă nu ajunge într-o poziție a Jucătorului 1, atunci se va alege o săgeată care duce în poziția adversarului cu cea mai mică valoare.

Această strategie poate fi îmbunătățită dacă se "privește în viitor" cu mai mulți pași. Totuși, găsirea strategiei optime poate fi găsită doar prin căutarea tuturor drumurilor posibile până la atingerea poziției inițiale.

Algoritmul probabilist

O altă posibilitate de rezolvare a problemei o constituie un algoritm care încearcă să ghicească săgeata corectă. Probabilitatea de a câștiga jocul alegând la întâmplare (aleator) săgețile este foarte mică.

P060105: Criptare dublă

Tom Verhoeff

Criptarea dublă, așa cum este ea aplicată în această problemă, oferă o securitate suplimentară mult mai mică decât s-ar părea la prima vedere. Dacă lungimea cheii este de n biți, atunci o căutare exhaustivă trebuie să ia în considerare toate cele 2^n chei. S-ar putea crede că criptarea dublă, folosind două chei independente a câte n biți, corespunde unei criptări simple cu o cheie având lungimea de $2 \cdot n$ biți. Totuși așa-numitul ***meet-in-the-middle attack*** (*atac cu întâlnire la mijloc*) arată că această criptare nu oferă o securitate mai mare decât o criptare simplă cu o cheie de lungime $n + 1$. În practică este folosită criptarea triplă, cunoscută sub numele de *Triple DES*.

Atacul cu întâlnire la mijloc funcționează astfel: textul p este criptat folosind toate cele 2^n chei posibile și rezultatele sunt stocate (folosind o tehnică oarecare) într-un tabel; în continuare, textul obținut după criptarea dublă este decriptat folosind toate cele 2^n chei posibile; fiecare rezultat al decriptării este căutat în tabel și în cazul în care s-a întâlnit o concordanță, atunci cheia k_2 este cheia curentă, iar cheia k_1 este cheia corespunzătoare poziției din tabel la care s-a găsit concordanța. Teoretic, este posibilă apariția mai multor concordanțe, dar este foarte puțin probabil să apară un astfel de caz. Pentru această problemă este suficientă găsirea unei singure perechi de chei.

Vom studia acum puțin mai detaliat efortul computațional de care este nevoie pentru găsirea celor două chei. Pentru a traversa spațiul cheilor posibile avem nevoie de două operații:

```
FirstKey(var key:Block);
NextKey(var key:Block):Boolean;
```

unde valoarea returnată de `NextKey` indică existența sau inexistența unui succesor pentru cheia curentă.

Structura de date corespunzătoare tabelului care păstrează rezultatele criptării textului p trebuie să pună la dispoziție următoarele operații:

```
EmptyTable;
Store(const msg:Block; const key:Block);
Retrieve(const msg:Block;
         var found:Boolean; var key:Block);
```

Operația `EmptyTable` inițializează un tabel vid.

Cu ajutorul operației `Store` este adăugată o pereche (`msg`, `key`) cu proprietatea `msg = Encrypt(p,key)`.

Prin intermediul operației `Retrieve` este verificată prezența unui mesaj `msg` și, în cazul găsirii unui astfel de mesaj, este furnizată valoarea cheii corespunzătoare `k` pentru care `msg = Encrypt(p,key)`.

Mai trebuie adăugată o operație care permite verificarea egalității a două mesaje (structuri de tip `Block`).

Operațiile `Store` și `Retrieve` trebuie să fie efectuate foarte rapid (preferabil în timp constant), fapt care sugerează folosirea unei tabele de dispersie (tabelă *hash*) deschise.

P060106: Depozit

Jyrki Nummenmaa, Erkki Mäkinen

Fără a restrânge generalitatea putem presupune că avem containere etichetate cu numere cuprinse între 1 și N.

Algoritmul de căutare exhaustivă

Vom genera toate permutările numerelor 1, 2, ..., N. Pentru fiecare permutare vom genera plasamentul containerelor în depozit folosind metoda muncitorului și vom compara plasamentul obținut cu plasamentul furnizat la intrare. Vom furniza la ieșire toate permutările cărora le corespunde un plasament identic cu cel de la intrare.

Totuși, un program care implementează această versiune a algoritmului nu se încadrează în limita de timp admisă decât pentru valori mici ale lui N.

Algoritm care folosește metoda backtracking

În continuare vom prezenta un algoritm bazat pe următoarea idee: vom elimina succesiv containere, determinând starea depozitului pentru situația în care containerul eliminat ar fi fost ultimul sosit în depozit. Determinarea stărilor după eliminarea unui container poate implica mutarea unor containere pe liniile de deasupra, operație opusă celei de la inserare (care implică mutări pe liniile de dedesubt).

Pentru fiecare container de pe prima linie vom apela metoda `RecursiveDelete`. Aceasta are parametrii `depot`, `item` și `solution`; pentru primul apel parametrul `solution` va fi un șir vid (nici un container).

Dacă `item` este singurul container rămas în depozit, atunci acesta este adăugat în lista `solution` și soluția obținută este furnizată la ieșire.

Dacă `item` nu este singurul container rămas în depozit, atunci acesta este adăugat în lista `solution`, este eliminat din depozit și pentru toate containerele de pe prima linie este apelată metoda `PullUp(depot,item,solution)`.

Metoda `PullUp` are trei parametri: `depot`, `item` și `solution`. Dacă ne aflăm pe ultima linie, atunci determinăm, pe baza valorilor corespunzătoare, toate containerele care pot înlocui containerul `item` din linia anterioară. Fiecare astfel de container va fi eliminat și pentru toate containerele `newitem` de pe prima linie vom apela `RecursiveDelete(depot,newitem,solution)`.

Acest algoritm ia în considerare toate posibilitățile de a elimina containerele din depozit, deci va determina toate soluțiile posibile ale problemei.

Din nefericire, acest algoritm nu este eficient. Sunt "favorizate" intrările care conțin linii scurte (în cazul cel mai favorabil avem N linii cu câte un container). În cazul în care avem o singură linie cu N containere, algoritmul nu este mai eficient decât cel care determină toate permutările posibile. Mai mult, datorită detaliilor de implementare, timpul de execuție este în unele cazuri mai mare.

O optimizare evidentă poate fi observată imediat. În momentul în care nu mai există containere decât pe prima linie, putem construi restul soluției în timp liniar, deoarece containerele respective trebuie să fi fost inserate în ordinea în care se află în acel moment. În caz contrar, inserarea unui container ar fi trebuit să ducă la mutarea pe linia următoare a unui alt container.

Chiar și folosind această optimizare, algoritmul nu este eficient. Dar, să observăm că `PullUp` execută mult mai multe operații decât ar fi necesar. Dacă studiem mai atent problema, "descoperim" lema prezentată în continuare.

Lemă

Presupunem că d este un depozit care conține N - 1 containere și în care intră un container. Considerăm operația de mutare a unui container y de pe linia r pe linia $r + 1$ și inserarea containerului x pe linia r, în locul containerului y. Este evident că avem $x < y$ și $y' < x$ pentru fiecare container y' aflat înaintea containerului y pe linia r înaintea mutării acestuia pe linia următoare.

Folosind această lemă ajungem la următorul rezultat:

Lemă

Presupunem că d este un depozit care conține N containere. Considerăm operația de mutare a unor containere de pe linia $r + 1$ pe linia r. În acest caz, fiecare container de pe linia $r + 1$ poate fi luat în considerare doar dacă este găsit înlocuitorul pentru exact un container de pe linia anterioară.

Folosind această lemă putem micșora foarte mult spațiul de căutare. Pentru a mări viteza de căutare putem păstra pentru fiecare container o listă cu containerele de pe linia următoare care pot fi mutate în locul său.

Dacă luăm în considerare următoarele rezultate, poate fi găsit un alt algoritm:

Lemă

În momentul inserării unui container, ultimul container care trebuie poziționat va ajunge pe ultima poziție a unei linii, iar linia de deasupra sa nu poate fi mai scurtă decât linia pe care se află.

Lemă

Despre fiecare container aflat la sfârșitul unei linii care nu este la fel de lungă ca linia de deasupra, se poate spune că a fost ultimul adăugat.

Aceste leme sugerează o căutare similară cu cea prezentată anterior, dar începând cu ultima linie. Avantajul principal îl constituie faptul că vom începe doar cu containere care pot fi mutate pe linia anterioară printr-o operație inversă inserării, spre deosebire de algoritmul anterior care începe căutarea de la prima linie și ar putea porni cu un container care nu poate fi ultimul container inserat.

Etapa FINALĂ

În numărul trecut am publicat enunțurile celor cinci probleme propuse spre rezolvare la faza finală a concursului de programare organizat de GInfo. Vă prezentăm acum descrierile soluțiilor oficiale. Implementările sunt disponibile pentru download la www.ginfo.ro/revista/11_7.

CPAR7P1: Bile

Problema poate fi rezolvată folosind metoda *programării dinamice*. Vom construi o matrice `cost`; elementul $cost_{ij}$ va indica lungimea minimă totală a barelor folosite pentru a lega bilele cuprinse între pozițiile i și j. În final, elementul $cost_{1,2n}$ va conține lungimea totală a barelor necesare pentru a lega toate bilele.

Este evident că toate elementele de pe diagonala principală vor avea valoarea `0`. Inițial vom considera că celelalte elemente (doar cele aflate deasupra diagonalei principale sunt semnificative) au valoarea ∞.

Pentru două bile vecine (aflate pe pozițiile i și $i+1$) care au culori diferite putem afirma că valoarea $cost_{i,i+1}$ este `3` deoarece avem nevoie de două bare verticale și una orizontală, toate având lungimea `1`.

Nu are rost să determinăm valorile corespunzătoare pozițiilor i și $i+k$, pentru care k este par, deoarece nu avem posibilitatea de a lega un număr impar de bile. Așadar, la pașii următori vom determina valorile corespunzătoare pozițiilor i și $i+k$, pentru care k este impar.

Există două posibilități de a determina valoarea minimă corespunzătoare pozițiilor i și j.

Dacă bilele au culori diferite, atunci putem să legăm cele două bile cu o bară orizontală de lungime $j-i$ și două verticale care au o înălțime cu `1` mai mare decât cea mai mare înălțime a barelor verticale de pe pozițiile cuprinse între $i+1$ și $j-1$.

Se observă că mai avem nevoie de o matrice (`height`), elementul $height_{ij}$ indicând înălțimea maximă a barelor verticale cuprinse între pozițiile i și j.

O altă posibilitate de obținere a unei valori pentru $cost_{ij}$ (indiferent de culoarea bilelor de pe pozițiile i și j) este de a alege o poziție intermediară k și a aduna valorile $cost_{ik}$ și $cost_{kj}$. Valoarea $height_{ij}$ va fi egală cu maximul valorilor $height_{ik}$ și $height_{kj}$. Vom alege toate valorile k pentru care $k-i$ este un număr par.

Evident, vom alege cea mai mică valoare pe care o determinăm folosind cele două posibilități.

Folosind această metodă vom completa toate elementele $cost_{ij}$ pentru care există o posibilitate de a lega toate bilele cuprinse între i și j. Pentru toate celelalte elemente se va păstra valoarea ∞.

După determinarea valorii minime trebuie reconstituite legăturile realizate. Pentru aceasta trebuie să determinăm modul în care au fost calculate valorile $cost_{ij}$. Va trebui să știm dacă am folosit prima sau a doua metodă și, în cazul celei de-a doua metode, care este valoarea k aleasă. Vom păstra încă o matrice (`inter`), iar elementele $inter_{ij}$ vor avea valoarea `-1` dacă a fost folosită prima posibilitate sau valoarea k dacă a fost folosită cea de-a doua.

Legăturile între pozițiile i și j se vor afișa printr-o rutină recursivă; inițial ea va fi apelată pentru pozițiile 1 și $2 \cdot n$.

În cazul în care se folosește prima posibilitate pentru pozițiile i și j, vom "tipări" legătura dintre i și j și apoi vom apela recursiv rutina pentru pozițiile $i+1$ și $j-1$.

În cazul în care se folosește cea de-a doua posibilitate pentru pozițiile i și j, nu vom tipări nimic și vom apela recursiv rutina pentru pozițiile i și k și pentru pozițiile $k+1$ și j.

CPAR7P2: Plăți și încasări

Enunțul problemei poate fi "tradus" astfel:

Se dă un șir de numere întregi, dintre care cel puțin unul este pozitiv. Să se determine subsecvența de sumă maximă a acestui șir. În cazul în care există mai multe astfel de subsecvențe trebuie determinată cea mai lungă dintre acestea. Dacă există mai multe subsecvențe de sumă maximă care au aceeași lungime, trebuie aleasă cea care începe cu elementul având indicele cel mai mic.

Această problemă (împreună cu soluția) a fost prezentată în articolul *Secvența de sumă maximă*, scris de ***Clara Ionescu*** și publicat în numărul 11/2 (februarie 2001) al revistei noastre.

CPAR7P3: Robot

Pentru a păstra traseul robotului vom folosi o matrice ale cărei elemente vor reprezenta pătrățelele caroiajului. Fiecare element va păstra informații referitoare la laturile pătrățelului corespunzător: existența unei linii deasupra, sub,

în stânga sau în dreapta pătrățelului. Cea mai simplă posibilitate de implementare este folosirea a patru biți, fiecare reprezentând una dintre cele patru laturi și având valoarea 1 dacă există linie pe acea latură și 0 în caz contrar.

După construirea matricei, trebuie determinat dreptunghiul vid de arie minimă care se formează. Pentru simplificare, vom trata patru cazuri:

1. dreptunghiuri de lungime și înălțime 1.
2. dreptunghiuri de lungime 1 și înălțime diferită de 1.
3. dreptunghiuri de lungime diferită de 1 și înălțime 1.
4. dreptunghiuri de lungime și înălțime diferite de 1.

Pentru primul caz va trebui doar să căutăm pătrățele care au linii pe toate cele patru laturi ale lor. Dacă găsim un astfel de pătrățel, atunci soluția problemei este un dreptunghi de arie 1.

Pentru al doilea caz va trebui să căutăm pătrățele care au linii pe trei dintre laturi (deasupra, sub și la stânga). Vom parcurge apoi pătrățelele din dreapta sa care au linii deasupra și dedesubt.

Când întâlnim primul pătrățel care nu respectă această condiție, e posibil să fi ajuns la ultimul pătrățel al dreptunghiului. Dacă acesta are linii deasupra, sub și la dreapta, atunci am găsit un nou dreptunghi. Dacă aria acestuia este mai mică decât cea a celui mai mic dreptunghi determinat anterior, vom păstra noua soluție.

Pentru al treilea caz va trebui să căutăm pătrățele care au linii pe trei dintre laturi (deasupra, la stânga și la dreapta). Vom parcurge apoi pătrățelele de sub el, care au linii la stânga și la dreapta.

Când întâlnim primul pătrățel care nu respectă această condiție, e posibil să fi ajuns la ultimul pătrățel al dreptunghiului. Dacă acesta are linii dedesubt, la stânga și la dreapta, atunci am găsit un nou dreptunghi. Dacă aria acestuia este mai mică decât cea a celui mai mic dreptunghi determinat anterior, vom păstra noua soluție.

Pentru al patrulea caz vom căuta pătrățele care au linii pe două dintre laturi (la stânga și deasupra). Vom parcurge apoi pătrățelele din dreapta sa care au linii numai deasupra.

Când întâlnim un pătrățel care nu respectă această condiție, e posibil să fi ajuns la latura din dreapta a dreptunghiului. Acest pătrățel trebuie să aibă linii pe latura de deasupra și pe cea din dreapta. Vom parcurge acum pătrățelele aflate sub pătrățelul inițial, care au linii numai la stânga.

Când întâlnim un pătrățel care nu respectă această condiție, e posibil să fi ajuns la latura de jos a dreptunghiului. Acest pătrățel trebuie să aibă linii pe latura de dedesubt și pe cea din stânga. Dacă toate condițiile au fost îndeplinite, am reușit să determinăm un posibil dreptunghi.

În plus, trebuie să verificăm dacă acesta este unul valid. Pentru aceasta vom parcurge pătrățelele de pe latura de jos și vom verifica dacă nu au și alte linii cu excepția celei de dedesubt.

Analog, pătrățelele de pe latura din dreapta trebuie să aibă linii numai pe latura din dreapta. În sfârșit, pătrățelul din colțul dreapta-jos trebuie să aibă linii pe latura din dreapta și pe cea de dedesubt.

Dacă și aceste condiții sunt îndeplinite, atunci am găsit un nou dreptunghi. Dacă aria acestuia este mai mică decât cea a celui mai mic dreptunghi determinat anterior, vom păstra noua soluție.

O observație importantă este că nu trebuie să verificăm decât pătrățelele de pe laturi, nu și pe cele din interior. Aceasta se datorează continuității liniei care formează traseul robotului. Pentru ca o linie să se afle în interiorul unui dreptunghi, aceasta ar fi trebuit să pornească din exteriorul acestuia, deci s-ar fi intersectat cu cel puțin una dintre laturi și dreptunghiul nu ar mai fi fost valid.

CPAR7P4: Star Trek

Este evident că pentru ca cercurile în care pot acționa *tricorder*-ele să fie disjuncte, trebuie să alegem cei mai apropiați doi membri ai echipajului.

După determinarea acestei perechi, raza maximă de acțiune poate fi calculată înjumătățind distanța dintre ele. Deoarece raza de acțiune trebuie să fie un număr întreg, valoarea obținută va fi trunchiată.

Observăm că, dacă razele sunt egale cu jumătatea distanței, cele două cercuri sunt tangente. Din acest motiv, în cazul în care valoarea obținută este un număr întreg, va trebui să mai scădem o unitate (acest caz apare în exemplul din enunț).

Așadar, problema se reduce la determinarea celei mai apropiate perechi de puncte cunoscând coordonatele a n puncte din plan. Această problemă a fost propusă spre rezolvare la etapa a doua a ediției trecute a concursului nostru. Enunțul a apărut în numărul 10/7 (noiembrie 2000) al *GInfo*, iar soluția în numărul 11/1 (ianuarie 2001).

CPAR7P5: Windows XP

Această problemă poate fi rezolvată folosind o strategie de tip *greedy*.

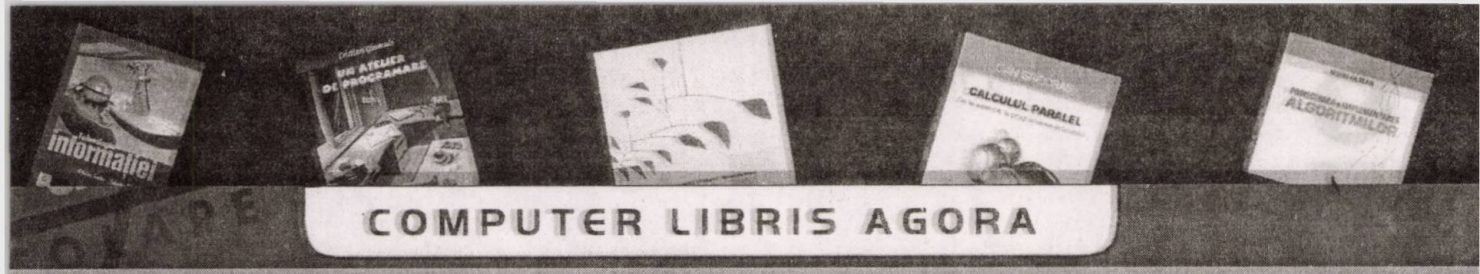
Pentru început vom crea o "listă de așteptare" pentru fiecare aplicație, listă care va conține toate aplicațiile care trebuie instalate în prealabil pentru ca programul de instalare să poată fi lansat în execuție.

Este evident că vom începe instalarea tuturor aplicațiilor care nu necesită instalarea prealabilă a altor aplicații.

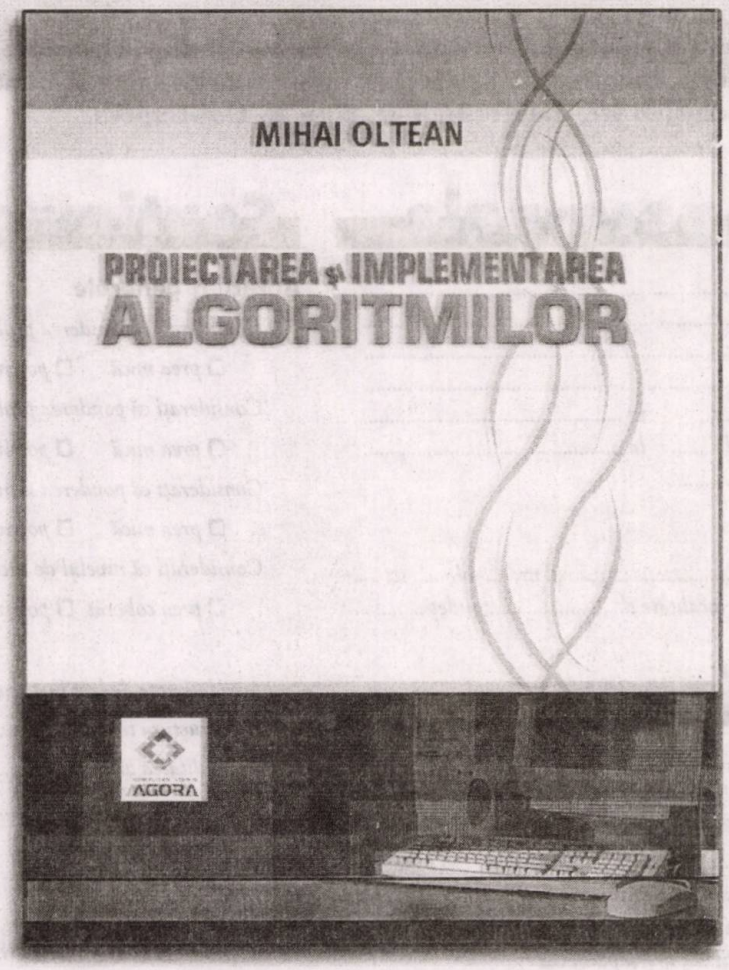
Apoi, la fiecare pas, vom determina toate aplicațiile al căror proces de instalare s-a încheiat și le vom elimina din listele de așteptare ale aplicațiilor care nu au fost încă instalate.

După aceea vom verifica dacă există aplicații pentru care listele de așteptare sunt vide. Dacă găsim astfel de aplicații putem să lansăm în execuție programul de instalare corespunzător.

La fiecare pas vom memora aplicațiile ale căror programe de instalare se află în execuție. În final vom afișa numărul de pași parcurși și, corespunzător fiecărui pas, lista aplicațiilor al căror proces de instalare este în desfășurare.



COMPUTER LIBRIS AGORA



"Cum aș putea să justific existența acestei cărți? Cum aș putea spre exemplu să justfic cele 100 de pagini despre programarea dinamică, acum când această metodă nu mai figurează în programele școlare? Nu știu. Sau, cum aș putea să justific capitolul despre algoritmi genetici, când pe plan mondial abia de câțiva ani a fost dovedită utilitatea și eficiența lor? Din nou nu știu! Dar rămâne bănuiala: a scrie algoritmi înseamnă a crea. Și nu poți crea fără să gândești."

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Rundele 2, 3, 4, 5!

Talon de PARTICIPARE

Acest talon dă dreptul de participare la Concursul de Programare Agora începând cu a doua rundă. Completați integral acest talon și trimiteți-l pe adresa redacției: str. Universității nr. 8 ap. 8, Cluj-Napoca.

Date personale

Nume: .....
Prenume: .....
Telefon: .....
Adresă e-mail: .....
❒ *elev în clasa* ..... *la* .....
❒ *student în anul* ..... *la* .....
Data nașterii: .....

Adresa

str. ..... *nr.* .... *bl.* ..... *sc.* .....
apt. ..... *cod* ..... *localitate* ..... *județ* .....

Chestionar

Folosiți un calculator acasă? ❒ *Da* ❒ *Nu*
Aveți acces e-mail acasă? ❒ *Da* ❒ *Nu*
Aveți acces WWW acasă? ❒ *Da* ❒ *Nu*
Plănuiți un upgrade anul acesta? ❒ *Da* ❒ *Nu*
Cât timp petreceți în fața calculatorului? ..... *ore/săptămână*
Cât timp petreceți online? ..... *ore/săptămână*
Cât timp vă jucați? ..... *ore/săptămână*
Cât timp programați? ..... *ore/săptămână*
Câte calculatoare sunt la școală/facultate? ..... *buc.*
Cum apreciați dotarea cu tehnică de calcul a școlii?
❒ *Dezastruoasă* ❒ *Foarte slabă* ❒ *Slabă*
❒ *Nesatisfăcătoare* ❒ *Medie* ❒ *Satisfăcătoare*
❒ *Bună* ❒ *Foarte bună* ❒ *Excelentă*
Considerați că prețul revistei este:
❒ *prea mic* ❒ *accesibil* ❒ *prea mare*
Ați dori ca revista să conțină și un CD? ❒ *Da* ❒ *Nu*
Ați dori ca revista să fie color? ❒ *Da* ❒ *Nu*
De cât timp cunoașteți GInfo? ..... *ani*
Cum ați procurat acest număr?
❒ *chioșc* ❒ *abonament* ❒ *cadou* ❒ *altfel:* .....
Ați dori ca GInfo să aibă:
❒ *4 numere/an* ❒ *8 numere/an*
❒ *12 numere/an* ❒ *24 numere/an*

Conținutul GInfo

Aprecieri generale

Considerați că ponderea problemelor rezolvate este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că ponderea problemelor propuse este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că ponderea subiectelor teoretice este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că nivelul de accesibilitate al articolelor este:
❒ *prea coborât* ❒ *potrivit* ❒ *prea ridicat*

Acordați note de la 1 la 10 pe următoarele criterii:
Structura tematică: .....
Calitatea articolelor: .....
Prezentare grafică: .....
Calitate tipografică: .....

În viitor

În următoarele numere așteptați:
mai multe articole de programare? ❒ *Da* ❒ *Nu*
mai multe articole teoretice? ❒ *Da* ❒ *Nu*
mai multe articole despre Internet? ❒ *Da* ❒ *Nu*
mai multe prezentări de programe? ❒ *Da* ❒ *Nu*
prezentări de jocuri? ❒ *Da* ❒ *Nu*
articole despre hardware? ❒ *Da* ❒ *Nu*

V-ar plăcea să citiți:
pagini redactate de elevii unor licee? ❒ *Da* ❒ *Nu*
lucrări premiate, realizate de elevi? ❒ *Da* ❒ *Nu*
actualități școlare? ❒ *Da* ❒ *Nu*
prezentarea unor situri Web? ❒ *Da* ❒ *Nu*
tips & tricks? ❒ *Da* ❒ *Nu*

Calculatorul

Indicați caracteristicile calculatorului pe care îl veți folosi la rezolvarea problemelor de la concurs.

Tipul procesorului

- ❒ *Intel Pentium*
- ❒ *Intel Pentium Pro*
- ❒ *Intel Pentium II*
- ❒ *Intel Pentium III*
- ❒ *Intel Pentium 4*
- ❒ *Intel Celeron*
- ❒ *AMD K6*
- ❒ *AMD K6-2*
- ❒ *AMD K6-III*
- ❒ *AMD Athlon*
- ❒ *AMD Duron*
- ❒ *Altul:* .....

Frecvența procesorului: ..... MHz

Tipul plăcii de bază: .....

Frecvența magistralei sistem: ..... MHz

Capacitatea memoriei sistem: ..... MB

Tipul plăcii video: .....

Capacitatea memoriei plăcii video: ..... MB

Tipul discului hard: .....

Capacitatea discului hard: ..... GB

Componente suplimentare

Unitate CD-ROM: ❒ Da ❒ Nu
Tip: .....
Viteza de citire: ..... ×

Unitate CD-RW: ❒ Da ❒ Nu
Tip: .....
Viteza de citire: ..... ×
Viteza de inscripționare: ..... ×
Viteza de reinscripționare: ..... ×

Placă de sunet: ❒ Da ❒ Nu
Tip: .....

Boxe active: ❒ Da ❒ Nu
Tip: .....

Placă de rețea: ❒ Da ❒ Nu
Tip: .....

Modem: ❒ Da ❒ Nu
Tip: .....
Viteza de transfer: ..... Kbps

Sistemul de operare

- ❒ *MS-DOS*
- ❒ *Microsoft Windows '95 OSR 2*
- ❒ *Microsoft Windows '98 Second Edition*
- ❒ *Microsoft Windows NT 4.0*
- ❒ *Microsoft Windows XP*
- ❒ *Microsoft Windows '95*
- ❒ *Microsoft Windows '98*
- ❒ *Microsoft Windows Me (Millennium Edition)*
- ❒ *Microsoft Windows 2000*
- ❒ *Altul:* .....

Aplicații folosite frecvent

Marcați aplicațiile instalate pe calculator și indicați versiunile corespunzătoare

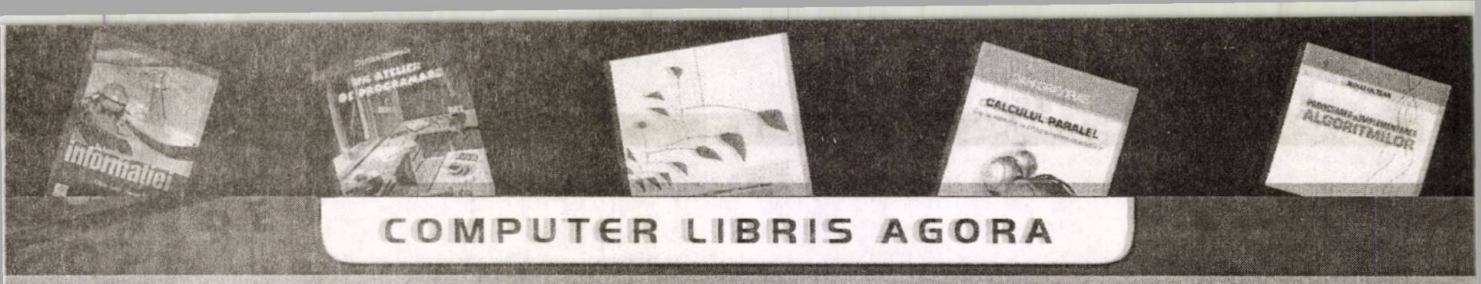
- ❒ *Microsoft Internet Explorer (versiunea .....)*
- ❒ *Microsoft Office (versiunea .....)*
- ❒ *Microsoft Visual Studio (versiunea .....)*
- ❒ *Borland Pascal (versiunea .....)*
- ❒ *Corel Draw (versiunea .....)*
- ❒ *Borland C++Builder (versiunea .....)*
- ❒ *Netscape Navigator (versiunea .....)*
- ❒ *Adobe Photoshop (versiunea .....)*
- ❒ *Adobe Acrobat Reader (versiunea .....)*
- ❒ *Borland C++ (versiunea .....)*
- ❒ *Borland Delphi (versiunea .....)*
- ❒ *Borland JBuilder (versiunea .....)*

Programe antivirus utilizate: .....

Arhivatoare utilizate: .....

Jocurile preferate: .....

Site-urile preferate: .....



COMPUTER LIBRIS AGORA

Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

AGORA

Bursele AGORA (Ediția 2001/2002)

În acest număr sunt publicate enunțurile celor trei probleme ale celei de-a doua runde. Dacă nu v-ați înscris încă, puteți participa începând cu această rundă. Pentru informații referitoare la concurs puteți vizita site-ul oficial al concursului la adresa www.ginfo.ro/concurs. Aici veți putea găsi enunțurile problemelor, clasamente parțiale, soluții oficiale, date de test folosite, anunțuri generale etc. Vă dorim mult succes în rezolvarea problemelor de la această rundă!

Participare

Pentru a participa la concursul *Bursele Agora* începând cu runda a doua vă rugăm să completați integral talonul destinat acestui scop și să îl trimiteți până la data de ***15 decembrie 2001*** (data poștei). Talonele care vor sosi la redacție după această dată vor da dreptul de participare doar începând cu runda a treia. Talonele incomplete nu vor fi luate în considerare. Redacția *GInfo* nu își asumă nici o răspundere pentru talonele care, din diferite motive, nu ajung în timp util la redacție.

În termen de ***cinci*** zile de la primirea talonului de către redacție, vă vom trimite prin e-mail codul dumneavoastră de identificare.

Concurenții care au primit deja codul de identificare nu trebuie să trimită și acest talon.

Vă rugăm să citiți cu atenție noul regulament. Există câteva modificări minore care constau, în special, în modalitatea de trimitere a soluțiilor și cea de înscriere.

Runda a doua

Cele trei probleme ale rundei a doua sunt *Matrice*, *Vânzări* și *Funcție*.

Prima dintre ele a fost propusă de ***Mihai Stroe***, student în anul IV la *Universitatea Politehnica* din București, fost olimpic internațional, câștigător al primei ediții a concursului nostru și clasat pe locul al doilea la cea de-a doua.

A doua problemă a fost propusă de către ***Mihai Scorțaru***, student în anul IV la *Universitatea Tehnică* din Cluj, redactor-șef adjunct al revistei noastre și organizator principal al concursului.

Ultima problemă a fost propusă de către ***Mihai Oltean***, doctorand și master al *Universității Babeș-Bolyai* din Cluj, autor al mai multor cărți de specialitate, printre care și câteva culegeri de probleme.

Rezolvările celor trei probleme pot fi trimise până pe data de ***31 decembrie 2001***. Rezultatele primei runde vor deveni publice pe data de ***15 ianuarie 2001***.

Pentru a evita întârzierile, vă sfătuim să trimiteți soluțiile cu câteva ore sau zile înaintea expirării termenului limită de trimitere a acestora.

Restricții și precizări

Nu se vor accepta rezolvări care ajung la redacție după termenul limită specificat, chiar dacă acesta este depășit doar cu câteva minute.

Redacția nu va trimite prin e-mail enunțurile problemelor, indiferent de motivul pentru care un concurent ar putea solicita acest lucru. Termenul limită de trimitere a problemelor este ultima zi a lunii (până la ora 24:00).

Organizatorii concursului își rezervă dreptul de a modifica termenele de publicare a problemelor, a soluțiilor sau a rezultatelor și a termenelor de trimitere a soluțiilor.

De asemenea, unele prevederi ale regulamentului pot fi eliminate, modificate sau pot fi adăugate altele noi. Acestea vor fi publicate în paginile revistei și vor intra în vigoare începând cu runda pentru care au fost publicate enunțurile problemelor în același număr al revistei.

Pentru eventualele contestații, vă rugăm să folosiți adresa *e-mail* a concursului. Vă recomandăm ca înainte de trimiterea unei contestații să verificați punctajul obținut folosind datele de test care sunt publicate pe *site*-ul oficial al concursului.

Pentru detalii suplimentare referitoare la concurs, vă rugăm să contactați membrii redacției *GInfo*.

Runda 2, problema 1

MATRICE

Denumirea subdirectorului: **MATRIX**

Denumirea fișierului de intrare: **MATRIX.IN**

Denumirea fișierului de ieșire: **MATRIX.OUT**

Descrierea problemei

Se consideră două matrice pătratice binare (valorile elementelor pot fi 0 sau 1) A și B care au N linii și N coloane și matricea $M_0 = [1]$, o matrice de dimensiuni 1×1 al cărei unic element are valoarea 1.

Regula matricei pentru o matrice M de dimensiuni $k \times k$ se aplică astfel:

- fiecare element din matricea M care are valoarea 1 este înlocuit cu matricea A;
- fiecare element din matricea M care are valoarea 0 este înlocuit cu matricea B;

Astfel se obține o matrice de dimensiuni $k \cdot N \times k \cdot N$.

Regula matricei va fi folosită de 7 ori astfel:

- la început, ea este aplicată asupra matricei M_0 și duce la obținerea matricei $M_1 = A$;
- la al doilea pas, ea este aplicată asupra matricei $M_1 = A$ și se obține matricea M_2;
- la al treilea pas, este aplicată asupra matricei M_2 și se obține matricea M_3;
- la următorul pas, este aplicată asupra matricei M_3 și se obține matricea M_4;
- urmează aplicarea regulii asupra matricei M_4 și obținerea matricei M_5;
- după aplicarea regulii asupra matricei M_5 se obține matricea M_6;
- la ultimul pas, regula matricei este aplicată asupra matricei M_6 și se obține matricea M_7.

Sarcina voastră este de a determina poziția celui de-al x-lea element cu valoarea 1 din matricea M_7. Pentru a determina poziția celui de-al x-lea element cu valoarea 1, veți parcurge matricea M_7 linie cu linie (de la prima până la ultima) și veți citi fiecare linie de la stânga la dreapta.

Date de intrare

Prima linie a fișierului de intrare va conține un număr întreg N, care va indica dimensiunile matricelor A și B.

Fiecare dintre următoarele N linii va conține câte N valori binare (0 sau 1) separate printr-un singur spațiu, reprezentând cele $N \cdot N$ elemente ale matricei A. Pe prima dintre aceste linii se vor afla elementele corespunzătoare primei linii a matricei, pe cea de-a doua linie cele corespunzătoare celei de-a doua linii a matricei etc. Primul element dintr-o linie va corespunde elementului de pe prima coloană a liniei respective, al doilea va corespunde elementului de pe a doua coloană etc.

Fiecare dintre următoarele N linii va conține câte N valori binare (0 sau 1) separate printr-un singur spațiu, reprezentând cele $N \cdot N$ elemente ale matricei B. Pozițiile elementelor sunt identice cu cele precizate pentru matricea A.

Pe ultima linie a fișiereului de intrare se va afla numărul natural x.

Date de ieșire

Fișierul de ieșire va conține o singură linie pe care se vor afla două numere întregi, separate printr-un spațiu, care vor indica linia și coloana în care se află cel de-al x-lea element cu valoarea 1 din matricea M_7.

Restricții

- $2 \le N \le 5$;
- Se garantează faptul că matricea are cel puțin x elemente cu valoarea 1.

Exemplu

MATRIX.IN

```
2
1 1
1 1
0 1
1 0
131
```

MATRIX.OUT

```
2 3
```

Timp de execuție/test: **1 secundă**

VÂNZĂRI

Denumirea subdirectorului: **SELLS**

Denumirea fișierului de intrare: **SELLS.IN**

Denumirea fișierului de ieșire: **SELLS.OUT**

Descrierea problemei

Compania *GInfo International* furnizează produse pentru N magazine. Inițial nici un magazin nu a cumpărat nici un produs. Cele N magazine sunt etichetate folosind numere cuprinse între 1 și N.

O operațiune de vânzare are loc atunci când unul dintre cele N magazine cumpără produse în valoare de x\$.

Pe măsură ce operațiunile de vânzare se desfășoară, directorul companiei dorește să primească informații referitoare la vânzări. Astfel, el poate cere comunicarea sumei totale care ar trebui încasată de la magazinele cuprinse între pozițiile i și j.

O operațiune de vânzare este descrisă prin trei numere naturale. Primul dintre ele este întotdeauna 1 și indică faptul că are loc o operațiune de vânzare. Al doilea este un număr cuprins între 1 și N și identifică magazinul căruia i se vând produse. Al treilea număr este x și indică valoarea vânzării.

O cerere de informații este descrisă tot prin trei numere naturale. Primul dintre ele este întotdeauna 2 și indică faptul că are loc o cerere de informații. Următoarele două sunt cuprinse între 1 și N și identifică magazinele i și j. În urma acestei cereri de informații, trebuie determinată suma vânzărilor către magazinele i, $i + 1$, ..., $j - 1$, j.

Date de intrare

Prima linie a fișierului de intrare conține numerele N și K, separate printr-un singur spațiu. Primul indică numărul de magazine, iar al doilea numărul total de operațiuni de vânzare și cereri de infomații.

Următoarele K linii conțin câte trei numere naturale, separate prin spații. Primul dintre aceste numere este t, și poate avea valoarea 1 sau valoarea 2.

Dacă valoarea numărului t este 1, atunci următoarele două numere i și x vor indica faptul că are loc o vânzare către magazinul i, în valoare de x\$.

Dacă valoarea numărului t este 2, atunci următoarele două numere i și j vor indica faptul că se cere o statistică a vânzărilor către magazinele cuprinse între pozițiile i și j. Vor fi luate în considerare doar vânzările descrise pe liniile anterioare liniei curente și nu și cele care vor fi descrise pe liniile următoare.

Date de ieșire

Fișierul de ieșire va conține câte o linie pentru fiecare dintre liniile din fișierul de intrare pentru care valoarea t este 2. Pe fiecare dintre aceste linii se va afla câte un număr care va indica rezultatul statisticii cerute de director.

Restricții și precizări

- $1 \leq N \leq 70000$;
- $1 \leq K \leq 200000$;
- suma totală a vânzărilor nu poate depăși 2000000000;
- pot fi efectuate mai multe vânzări către același magazin.

Exemplu

SELLS.IN

```
7 9
1 3 1000
1 5 2000
2 4 6
1 4 3000
2 4 6
2 1 4
2 1 2
1 1 4000
2 1 7
```

SELLS.OUT

```
2000
5000
4000
0
10000
```

Timp de execuție/test: **3 secunde**

Runda 2, problema 3

FUNCȚIE

Denumirea subdirectorului: **FUNCTION**

Denumirea fișierului de intrare: **FUNCTION.IN**

Denumirea fișierului de ieșire: **FUNCTION.OUT**

Descrierea problemei

Se consideră o funcție $f(x)$ și se cere determinarea tuturor rădăcinilor reale situate în intervalul $[a, b]$ ale ecuației $f(x) = 0$.

Programul vostru nu va cunoaște descrierea exactă a acestei funcții, în schimb ea va putea fi apelată prin intermediul unui modul extern.

Programatorii în *Pascal* vor avea la dispoziție un unit numit `FUNC.PAS`. Acesta va conține doar funcția `f()`. Funcția va fi declarată astfel:

```
function f(x:Real):Real;
begin
  f:=... { descrierea functiei }
end;
```

Programatorii în *C/C++* vor avea la dispoziție un fișier *header* numit `FUNC.H`. Acesta va conține doar funcția `f()`. Funcția va fi declarată astfel:

```
float f(float x){
  return ... // descrierea functiei
}
```

În timpul evaluării, în teste diferite această funcție va fi diferită.

Date de intrare

Prima linie a fișierului de intrare conține numerele reale a și b, separate printr-un singur spațiu. Acestea reprezintă extremitățile intervalului în care se caută soluțiile.

Date de ieșire

Fișierul de ieșire va conține pe prima linie un număr întreg N care reprezintă numărul soluțiilor găsite.

Fiecare dintre următoarele N linii vor conține câte un număr real care reprezintă o soluție a ecuației (cu cinci zecimale exacte).

În cazul în care există două soluții x_1 și x_2 care, afișate cu cinci zecimale exacte, sunt identice atunci numai una dintre ele va fi luată în considerare.

Un număr x este considerat soluție a ecuației $f(x)$ dacă $|f(x)| < 0.00001$.

Restricții

- Ecuația are cel mult 20 de soluții în intervalul specificat.
- Funcția este continuă pe intervalul specificat.
- $-10000 \le a < b \le 10000$;

Exemplu

Presupunem că funcția necunoscută este $\sin x$.

`FUNCTION.IN`

```
-4 5
```

`FUNCTION.OUT`

```
-3.14159
0
3.14159
```

Modul de punctare

Vom presupune că unui test îi corespunde un punctaj maxim de p puncte.

În cazul în care una dintre soluțiile determinate de programul vostru este incorectă, sau există soluții care apar de mai multe ori, nu veți primi nici un punct pentru testul respectiv.

În cazul în care pogramul determină toate cele N soluții ale ecuației veți primi întregul punctaj.

În cazul în care nu ați determinat toate soluțiile posibile, dar lipsesc cel mult jumătate dintre acestea, veți primi $[2 \cdot p / 3]$ puncte.

În cazul în care nu ați determinat toate soluțiile posibile, dar lipsesc mai mult de jumătate dintre acestea, veți primi $[p / 3]$ puncte.

Timp de execuție/test: **3 secunde**

PERCEPTRONUL cu un singur strat

Lucian Burja

Un calculator ce simulează gândirea umană este un vis vechi care în prezent nu poate fi încă realizat. Până și cele mai performante calculatoare și programe software sunt departe de a reproduce credibil procesele cognitive care se desfășoară la nivelul creierului. Dar un fapt îmbucurător este că bazele în acest sens au fost puse, sub forma rețelelor neuronale.

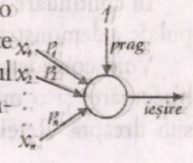
Rețelele neuronale încearcă să simuleze structura neurofiziologică a creierului uman. Creierul este alcătuit dintr-o mulțime de neuroni interconectați. Fiecare neuron primește informații de la alți neuroni prin intermediul dendritelor și transmite la rândul său un semnal prin intermediul axonului. Acest model poate fi implementat pe calculator, considerând legăturile dintre neuroni numere (ponderi) care se modifică în funcție de cât de des este activată legătura.

Primul model de rețea neuronală a apărut în 1958, când ***Frank Rosenblatt*** publică prima sa lucrare despre perceptron, denumire dată rețelei sale neuronale. Lumea oamenilor de știință a rămas fascinată de posibilitățile pe care le ofereau rețelele neuronale, care puteau fi folosite la recunoașterea unor structuri specifice electrocardiogramelor (recunoaștere de formă am spune astăzi).

Dar în 1969, ***Marvin Minsky*** și ***S. Papert*** au analizat lucid posibilitățile de învățare ale perceptronului și au ajuns la concluzii destul de sceptice, dovedind imposibilitatea rezolvării de către perceptronul cu un singur strat a unor probleme simple, cum ar fi învățarea funcției XOR. Ca urmare, interesul pentru acest domeniu de cercetare a scăzut dramatic. Relansarea interesului oamenilor de știință s-a produs în 1986, odată cu apariția unor modele și tehnici noi de instruire (metodele propagării înapoi, de exemplu).

În acest articol ne vom ocupa de cea mai simplă rețea neuronală posibilă, perceptronul cu un singur strat (având un singur neuron).

Neuronul are mai multe intrări și o singură ieșire. Valorile pe care le poate avea ieșirea sunt +1 și -1, deci neuronul poate realiza o clasificare a datelor de intrare în două categorii.



El posedă și un prag de activare, adică ieșirea sa poate fi +1 sau -1 în funcție de semnalul de intrare și de valoarea *prag*. Dacă semnalul de intrare depășește valoarea *prag*, atunci ieșirea va fi +1, în caz contrar ea fiind -1. Semnalul de intrare se calculează folosind o sumă ponderată a valorilor de intrare:

$$I' = \sum_{i=1}^{n} x_i \cdot p_i$$

unde:

- $x_1, x_2, \ldots, x_n$ sunt datele de intrare;
- $p_1, p_2, \ldots, p_n$ sunt ponderile legăturilor;
- n este numărul intrărilor.

Dacă $I' > prag$, ieșirea va avea valoarea +1, iar în caz contrar va avea valoarea -1.

Pentru simplitate, putem considera pragul ca fiind o intrare separată care are întotdeauna valoarea semnalului de intrare egală cu +1 și ponderea egală cu pragul. În acest caz semnalul de intrare al neuronului va fi $I = I' - prag$ și avem condiția de activare $I > 0$. Avantajul acestei abordări este acela că valoarea *prag* poate fi și ea modificată în procesul de instruire, împreună cu celelalte ponderi.

Procesul de instruire este cel care permite perceptronului să poată clasifica datele de intrare. Prin acest proces, perceptronul "învață" și aici găsim secretul succesului pe care îl au rețelele neuronale. De multe ori, în practică, este dificil de implementat un algoritm de rezolvare a unei anumite probleme (recunoaștere de forme, de imagini, de voce etc.) sau rezultatele sunt nesatisfăcătoare. Rețelele neuronale nu au nevoie să cunoască nici un algoritm, ci pot fi antrenate folosindu-se un set de exemple, ceea ce le face deosebit de utile pentru aplicații de genul celor de mai sus. Performanțele obținute prin folosirea rețelelor sunt adesea excelente. De exemplu, s-a creat programul *TD Gammon*, care joacă table pe baza unei rețele neuronale la nivel de

maestru internațional. Pentru instruire, programul a pornit fără cunoștințe prealabile și a atins performanțe remarcabile după ce a jucat câteva sute de mii de partide contra lui însuși.

Revenind la perceptron, antrenarea acestuia se desfășoară în următorul mod: ponderile inițiale sunt generate aleator; apoi se prezintă rețelei date de intrare pentru care se cunoaște răspunsul dorit. Dacă răspunsul rețelei este diferit față de răspunsul dorit, atunci se ajustează ponderile, în așa fel încât perceptronul să clasifice corect datele de intrare. După ce rețeaua clasifică corect toate exemplele, putem considera că antrenarea s-a încheiat și ne putem aștepta ca în momentul introducerii unor seturi de date noi, perceptronul să răspundă corect.

Desigur, ne punem întrebarea dacă întotdeauna este posibil să găsim valori ale ponderilor, astfel încât perceptronul să clasifice corect datele de intrare. ***Teorema de convergență*** a perceptronului ne asigură că, pentru clase liniar separabile există întotdeauna un vector pondere astfel încât rețeaua să clasifice corect exemplele de instruire. Demonstrarea acestei teoreme presupune elemente de matematică superioară și poate fi găsită în resursele indicate în bibliografie. Prin ***clase liniar separabile***, înțelegem două clase ce pot fi despărțite printr-un hiperplan (o dreaptă în plan, un plan în spațiul 3D etc.) Acest concept va deveni mai clar puțin mai încolo, când vom apela la un exemplu. Pentru moment este suficient să reținem că ecuația hiperplanului de separație este furnizată de valorile ponderilor perceptronului.

O altă problemă care ne-o punem este cum să modificăm ponderile rețelei, în așa fel încât să ajungem la soluție, atunci când ea există.

Ideea de bază pentru antrenarea oricărui tip de rețea este folosirea unei ***funcții criteriu*** pe care încercăm să o minimizăm. În cazul nostru ***funcția criteriu*** este reprezentată de suma distanțelor până la hiperplanul de separație a punctelor greșit clasificate. Cu cât valoarea acestei funcții este mai mică (chiar zero), cu atât hiperplanul nostru separă mai bine cele două clase, deci este mai apropiat de soluție.

Minimizarea funcției criteriu se poate face prin ***metode de tip gradient***. Deoarece sunt necesare noțiuni de matematică ce depășesc programa de liceu, vom prezenta numai rezultatul final, demonstrația putând fi găsită în resursele indicate în bibliografie. Regula de corecție este:

$$p_{t+1} = p_t + \frac{c}{2} \cdot (d_t - o_t)$$

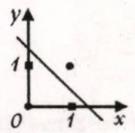
unde:

- p_{t+1} este vectorul de ponderi la pasul $t+1$;
- p_t este vectorul de ponderi la pasul t;
- d_t este rezultatul dorit la pasul t;
- o_t este rezultatul obținut la pasul t
- x_t este vectorul valorilor de intrare la pasul t;
- c este constanta de corecție $(0 < c \leq 1)$

Liniar separabilitatea este una din cele mai importante limitări ale perceptronului. De multe ori, exemplele de instruire în practică, nu constituie clase liniar separabile. Una dintre cele mai celebre probleme pe care un perceptron cu un singur strat nu o poate rezolva este problema funcției XOR. Tabelul de valori al funcției XOR este următorul:

x	y	x XOR y
1	1	0
1	0	1
0	1	1
0	0	0

Dacă reprezentăm funcția într-un sistem de axe, obținem următorul grafic:



unde prin cerculețe am notat valorile 0 ale funcției, iar prin pătrățele valorile 1. Se observă că oricum am duce o linie, nu vom reuși niciodată să separăm cele două clase de puncte.

O modificare a algoritmului clasic de instruire a fost adusă de către ***Gallant*** în anul 1986, astfel încât acesta să poată fi aplicat și claselor liniar inseparabile. Ideea este de a antrena rețeaua printr-un număr mare de pași, și de a reține configurația care a clasificat corect cele mai multe date de intrare. De multe ori însă, acest algoritm nu prezintă aplicabilitate practică, deoarece sunt necesare extrem de multe iterații pentru a ajunge la o soluție acceptabilă.

Un alt algoritm de instruire a perceptronului cu un singur strat este algoritmul ***Widrow-Hoff***, cunoscut și sub numele de ***regula delta***. Acest algoritm alege drept ***funcție criteriu*** suma erorilor pătratice:

$$\sum_{k=1}^{n} (d_k - o_k)$$

unde n reprezintă numărul exemplelor de antrenare, d_k ieșirea dorită pentru exemplul k, iar o_k rezultatul obținut. Se urmărește minimizarea ei, folosind din nou ***metode de tip gradient*** și se ajunge la următoarea regulă de instruire:

$$p_{t+1} = p_t + \frac{c}{t} \cdot \delta_t \cdot x$$

$$\delta_t = d_t - o_t$$

unde simbolurile au semnificația cunoscută.

În acest caz, rețelei i se prezintă exemple de instruire, până când eroarea de clasificare este mai mică decât o eroare acceptabilă, moment în care încheiem procesul de antrenare.

Am realizat implementările celor doi algoritmi prezentați anterior. Acestea sunt disponibile pentru download la adresa **www.ginfo.ro/revista/11_7**.

În continuare vom prezenta o aplicație simplă în scopul de a demonstra funcționalitatea perceptronului.

Vom considera o dreaptă în plan (de exemplu prima bisectoare) și o mulțime de puncte situate deasupra sau sub dreaptă. Rețelei îi vom prezenta câteva exemple de

puncte situate deasupra (ieșirea corectă +1) și dedesubt (ieșirea corectă -1), apoi o vom testa pentru puncte care nu sunt conținute în datele de antrenare. Datele de intrare se citesc din fișierul `PERCEPT.TXT`, unde pe prima linie avem numărul de puncte de antrenare, iar pe următoarele *n* linii triplete de numere ce reprezintă coordonatele punctului și răspunsul așteptat (+1 sau -1 în funcție de poziția punctului - deasupra sau sub linia considerată).

Programul declară două rețele care vor fi antrenate cu algoritmul clasic, respectiv cu *algoritmul delta*. Parametrul din apelul constructorului indică numărul de intrări ale rețelei (două intrări în cazul nostru, corespunzător celor două coordonate ale unui punct în plan). Antrenarea se face prin apelul funcției membru `Antreneaza()` a fiecărei clase. Se observă că această funcție este redefinită în clasa `Perceptron_Delta`, deoarece algoritmul de instruire este, în mod evident, altul. Răspunsul se obține prin apelarea funcției `Iesire()`.

Codul sursă al programului este disponibil pentru download la adresa **www.ginfo.ro/revista/11_7**.

Desigur, această problemă se poate extinde în spațiul tridimensional, sau chiar mai departe, prin mărirea numărului de intrări ale neuronului.

Se observă că, în general, *algoritmul de instruire* al lui *Widrow-Hoff* produce o soluție mai bună decât algoritmul clasic, deoarece avem posibilitatea să controlăm eroarea hiperplanului de separație. În cazul algoritmului clasic, instruirea se oprește în momentul în care toate exemplele de instruire au fost corect clasificate, chiar dacă soluția obținută este acceptabilă "la limită". Dacă clasele nu sunt liniar separabile, *algoritmul delta* va produce cu siguranță o soluție mai bună, deoarece algoritmul clasic va oscila și nu va converge niciodată, și chiar și în varianta *Gallant* va produce rezultate mai puțin satisfăcătoare.

Concluzie

Gama de probleme pe care un perceptron cu un singur strat le poate rezolva este destul de limitată, dar el reprezintă punctul de plecare pentru rețele mult mai complexe cu aplicații practice. Exemplul prezentat mai sus are scop pur didactic, dar considerăm că reușește să dea o imagine asupra puterii ce se află în spatele unei rețele neuronale. Dacă un singur neuron a reușit să rezolve problema de mai sus, putem ușor să ne închipuim ce pot face mai mulți neuroni organizați pe mai multe straturi...

Bibliografie

1. D. Dumitrescu, Hariton Costin, *Rețele neuronale. Teorie și aplicații*, Editura Teora, București, 1996;
2. **http://satirist.org/learn-game/systems/gammon/**;
3. **http://www.geocities.com/CapeCanaveral/1624/**;
4. **http://home.cc.umanitoba.ca/~umcorbe9/perceptron.html**;
5. **http://www.cs.bgu.ac.il/~omri/Perceptron/**.

Lucian Burja este student în anul II la Universitatea Politehnica din București. Poate fi contactat prin e-mail la **lucian_burja@email.ro**.

Tasta "magică"

Sistemele de operare de la *Microsoft* au implementate, pe lângă specificațiile din documentații, și alte funcții care au diferite roluri.

De exemplu, tasta *Shift* folosită în diferite combinații, modifică funcția care ar fi fost efectuată dacă această tastă nu ar fi fost apăsată. În continuare, vom prezenta câteva dintre opțiunile la care avem acces folosind tasta "magică".

Pentru a reporni interfața grafică *Windows '95/'98* (GUI) și o parte din *drivere*, efectuați un click pe butonul *Start*, selectați opțiunea *Shut Down* și, în fereastra de dialog selectați *Restart computer* și țineți apăsată tasta *Shift* în timp ce apăsați pe butonul *OK*.

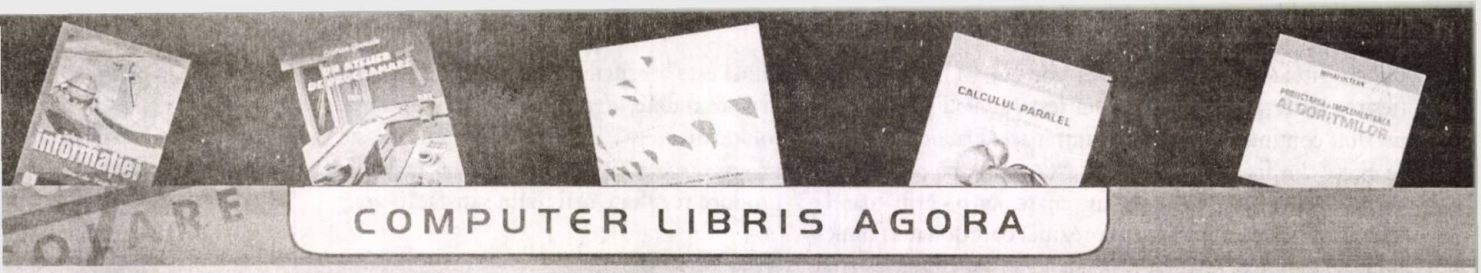
Majoritatea CD-urilor apărute în ultimul timp au un program care se execută la introducerea CD-ului în unitate. Pentru a evita pornirea acestui program, țineți apăsată tasta *Shift* din momentul în care introduceți CD-ul în unitate, până când acesta este accesat.

Dacă vreți să ștergeți unul sau mai multe fișiere selectate fără a le mai putea recupera (fără să mai treacă prin *Recycle Bin*) apăsați combinația de taste *Shift + Delete*.

Pentru a deschide un document folosind o altă aplicație (diferită de cea care este asociată cu tipul de document respectiv), țineți apăsată tasta *Shift* în timp ce efectuați un click dreapta pe fișierul selectat (sau apăsați *Shift + F10*, fără click de mouse). În acest fel va apărea opțiunea *Open With...* în meniul popup...

Dacă ați configurat *Explorer*-ul să deschidă pentru fiecare director o fereastră nouă, pentru a închide simultan toate ferestrele deschise, țineți apăsată tasta *Shift* în timp ce închideți ultima fereastră deschisă.

Pentru a intra în *Safe Mode*, tot tasta *Shift* este cea care vă salvează, chiar dacă meniul de boot este dezactivat. Mențineți tasta *Shift* apăsată în momentul încărcării sistemului de operare.



COMPUTER LIBRIS AGORA

Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.

Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Considerații asupra FRACTALILOR

Ciprian Ileană, Ioana-Maria Ileană

Teoria fractalilor și cea a haosului se numără printre cele mai intersante și mai noi ramuri ale matematicii. Cu o istorie relativ recentă (aproximativ 50-60 de ani) ele s-au impus repede în lumea științei, având aplicații în aproape toate domeniile, de la informatică până la geografie, economie, statistică etc.

În acest articol veți putea citi o introducere în teoria fractalilor și a haosului; aceasta este urmată de o prezentare a celor mai celebri fractali, printre care se regăsesc triunghiul lui *Sierpinski*, garnitura lui *Sierpinski*, covorul lui *Sierpinski*, curba lui *Peano* de umplere a spațiului, seriile *Mandelbrot* și seriile *Julia*. Va fi explicată noțiunea de dimensiune fractală și vor apărea definițiile unor termeni des folosiți.

Istoria fractalilor și a haosului

Geometria fractală și teoria haosului revoluționează matematica și știința, oferind o nouă cale de percepere a realității. S-au scris mai multe cărți despre știința haosului și geometria fractală, iar un număr de programe afișează fractali matematici cum ar fi faimoasa serie *Mandelbrot*.

De la artiști până la matematicieni, de la geografi până la astronomi, de la meteorologi până la experți financiari, efectiv nici un domeniu de activitate nu a rămas neatins de această modificare radicală în modul nostru de muncă, de gândire și de creație.

Mare parte din interesul de care se bucură fractalii este datorat faptului că matematica s-a reîntors la lumea macroscopică, la lumea palpabilă pe care o putem vedea cu toții, lăsând la o parte universul alambicat al tabelelor pline de cifre și litere grecești și al acceleratoarelor de particule.

Istoria în sine a fractalilor nu este lungă. Ea a început în anul 1875 odată cu apariția revoluționarei lucrări a matematicianului ***Benoit Mandelbrot***, "*O teorie a seriilor fractale*", lucrare care a devenit ulterior cartea sa manifest "*Geometria fractală a naturii*".

Cuvântul ***fractal*** a fost inventat de către *Mandelbrot*, el utilizându-l pentru a reuni munca multor altor matematicieni dinaintea sa. Printre primii matematicieni care au creat fractali (în general privindu-i doar ca pe niște exerciții abstracte, neavând nici o idee despre semnificațiile pe care le-ar putea avea aceștia) se numără nume cum ar fi: ***Waclav Sierpinski***, ***Helge von Koch***, ***Georg Cantor*** sau ***David Holbert***. Ei au simțit că descoperiseră ceva care sfida și amenința câteva dintre convingerile cele mai puternice și

Benoit Mandelbrot

Fondator al geometriei fractale, ***Benoit Mandelbrot*** s-a născut în *Polonia*, în anul 1924. În 1936 emigrează în *Franța* împreună cu întreaga familie și intră astfel sub influența unchiului său, profesor de matematică în *Paris* (*College de France*), matematician "de modă veche", riguros și exigent, succesor al marelui *Jacques Solomon Hadamard*. Astfel, prin intermediul acestui adevărat părinte spiritual, *Mandelbrot* ia contact atât cu matematica lui *Gaston Julia*, cât și cu "ariditatea" matematicii clasice. Contradicția dintre propria părere despre matematică și natura scolastică și rigidă impusă de matematicienii vremii a condus la germinarea și dezvoltarea unui drum nou: geometria fractală. Anul 1967 poate fi considerat anul de naștere al geometriei fractale, o dată cu apariția articolului "*How long is the coast of Britain?*", iar cartea "*The Fractals Geometry of Nature*" editată în 1982 consacra definitiv această noțiune ce devine practic domeniu al matematicii, fizicii și biologiei moderne.

Figura 1

mai prețioase ale matematicii acelor timpuri. În perioada 1875-1925 (cu aproximație), perioadă de mare criză în matematică, s-au descoperit forme bizare ce contraziceau conceptele cu privire la spațiu, suprafață, distanță sau dimensiune ale matematicienilor acelor timpuri.

Haos și sisteme dinamice

Dacă ar fi să analizăm lumea din jurul nostru am ajunge la o împărțire a lucrurilor în două mari clase: o clasă a *ordinii* și o alta a *dezordinii*. Din această cauză suntem poate tentați să asociem aceste două clase cu dihotomia: bine - rău, lucru destul de evident, dacă ne gândim că este de preferat pentru fiecare dintre noi să fim înconjurați de evenimente și lucruri care se desfășoară în timp și spațiu, urmând un tipar și o serie de reguli clare, cunoscute, care știm că nu se schimbă și nu sunt încălcate.

Din păcate, acest lucru nu este posibil în realitate; nu putem pretinde, de exemplu, unui artist ca, la intervale bine definite, să genereze doar opere de valoare *universală*. Am putea conchide că acolo unde lucrurile sunt clare, iar scopul sistemului este doar de a reproduce un flux tehnologic, de a reproduce o acțiune ce implică sincronizări de persoane, ordinea este imperios necesară. Într-un astfel de sistem, în care însăși structura și modul său de funcționare asigură predictibilitatea, orice variație este sinonimă accidentului.

La polul opus se află starea de dezordine, stare ce pare a fi cea normală în cazul oricărui tip de sistem, cel puțin până la o intervenție de factori coercitivi, structuranți care să încerce impunerea unei stări de ordine. În realitate, interacțiunile ce au loc între numeroasele *celule* ale unui sistem complex conduc în mod evident la o variabilitate a parametrilor măsurabili. Ca urmare, predicția nu este posibilă decât statistic, lucru ce nu ne prea ajută într-o situație dată. Un astfel de sistem în cadrul căruia subsistemele (celulele) se manifestă liber, după bunul lor plac, se numește *sistem haotic*.

Pentru cei neavizați, ideea unei teorii despre haos sau a unei metodologii cu privire la controlul haosului poate părea o aberație, ea fiind asociată fie conceptului de dezordine, fie unor termeni ce amintesc de procesul genezei din teologie. Această confuzie a apărut din păcate, în mare parte, datorită nefericitei alegeri a unei etichete pentru un domeniu de studiu fascinant. Pentru unii, acesta a fost chiar motivul pentru a aborda acest domeniu, în speranța obținerii unor informații transcendente, dar au fost surprinși dând peste o matematică *aridă* împânzintă de experimente cu ecuații diferențiale. Acest lucru i-a determinat pe mulți să abandoneze înainte de a descoperi extraordinarele aplicații practice care decurg din acest capitol al științei: comunicarea prin haos, stabilizarea sistemelor neliniare, structura haotică a unui semnal encefalografic și restabilirea haoticității normale a activității neuronale în tratamentele pentru epilepsie ș.a.m.d.

Am văzut până acum că unele concepte precum *haos* și *teoria haosului* pot părea ambigue. Pentru a le clarifica, vom arunca o privire asupra a ceea ce înseamnă complexa noțiune de *sistem dinamic*.

În general, printr-un sistem înțelegem o colecție de elemente și efectele lor unul asupra altuia. Aceasta pare destul de abstract. Dar, de fapt, suntem înconjurați de sisteme. Vremea, o pădure, economia globală, o mulțime de oameni pe un stadion de fotbal, peștii dintr-un lac, o centrală nucleară reprezintă exemple de sisteme, al căror *comportament* se poate schimba foarte rapid.

Sistemele reale își semnalează prezența prin trei factori:

- sunt dinamice, adică supuse schimbărilor;
- sunt complexe, adică depind de mulți parametri;
- sunt iterative, adică legile care le guvernează comportamentul pot fi descrise prin *feedback*.

Nimeni nu poate descrie complet prin formule matematice interacțiunile unui astfel de sistem. Cu toate acestea, se încearcă investigarea regularităților care formează baza unui astfel de sistem dinamic, în particular, găsirea unor modele matematice simple, cu ajutorul cărora să se poată simula comportamentul sistemului. Putem prezenta aceasta schematic ca în figura 2.

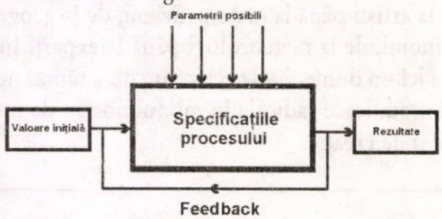


Figura 2: Schematica unui sistem haotic.

Termeni...

Haos: în limbajul curent, orice sistem care este înfricoșător prin incontrolabilitatea și dezordinea sa. Cuvântul a primit și un sens tehnic specific: dependența sensibilă de condițiile inițiale. Cu alte cuvinte, o mică schimbare în starea unui sistem haotic poate duce rapid la mari modificări în starea sistemului, măsurată la un moment ulterior (chiar dacă starea sistemului este complet dependentă de aceste condiții inițiale).

Calcul diferențial: ramură a matematicii care se ocupă cu găsirea pantei sau a raportului de schimbare a curbelor. Această pantă este denumită *diferențială* sau *linie tangentă*.

Diferențiere: Procedeul matematic de determinare a valorii pantei unei curbe într-un punct dat (vezi și *calcul diferențial*).

În general numim *haotic* un comportament imprevizibil, în contextul de mai sus. Cauza acestuia este bazată pe faptul că schimbări neglijabile asupra cantității grupate mai apoi de *feedback* pot produce efecte haotice neașteptate. Acest fenomen uimitor apare în particular într-o gamă de probleme care ar putea ridica semne de întrebare asupra unor teorii recunoscute, sau stimula noi formulări în biologie, fizică, matematică, chimie și în domeniul economic.

Așadar, aria de cercetare în teoria sistemelor dinamice este în mod manifest interdisciplinară. Această teorie este încă tânără și pare suficient de simplă matematic pentru ca oricine cu abilități elementare de programare și calculator să poată să-i aprecieze rezultatele. Scopul cercetării este de a înțelege cum are loc tranziția de la ordine la haos.

O posibilitate importantă pentru a investiga sensibilitatea sistemelor haotice este de a le reprezenta comportamentul prin grafică pe computer. Aceste forme grafice rezultate apar sub forma unor fractali.

Fractali celebri

În cele ce urmează vom prezenta o serie de fractali care au devenit celebri.

Triunghiul lui Sierpinski

Probabil cel mai cunoscut fractal al tuturor timpurilor este așa-numitul *triunghi al lui Sierpinski.*

Modul de realizare al acestui fractal este foarte simplu: la început se desenează un triunghi pe care îl vom diviza în patru părți egale, iar trei dintre ele (cele din exterior) vor fi și ele divizate (folosind același procedeu), procesul continuând la infinit pentru toate triunghiurile formate.

În figura 3 este prezentată modalitatea de construcție descrisă anterior.

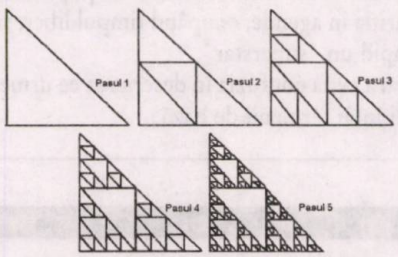


Figura 3: Triunghiul lui Sierpinski

Desigur, pentru a putea realiza forma matematică imaginată de *Sierpinski* ar însemna să putem trasa linii de divizare infinit de mici. În figură sunt ilustrați doar primii pași efectuați.

O altă modalitate de realizare a acestui fractal este următoarea: se pornește de la un triunghi plin în care se "decupează" triunghiuri egale cu un sfert din triunghiul inițial. Dacă repetăm la infinit acest proces, atunci vom obține un fractal identic cu *triunghiul lui Sierpinski* (uneori, când este realizat prin această tehnică el se mai numește și ***garnitura lui Sierpinski.*** Primii pași ai acestui procedeu sunt prezentați în figura 4.

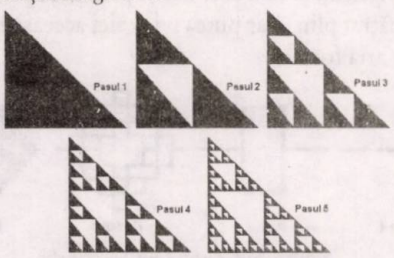


Figura 4: Garnitura lui Sierpinski

Până acum totul ar părea în regulă, nici o lege a matematicii nefiind încălcată. Problema a apărut atunci când *Sierpinski* a încercat să determine aria pe care o ocupă acest fractal. Pe de o parte am putea crede că aria este zero, deoarece un număr infinit de găuri acoperă în cele din urmă orice suprafață plină din triunghi. Pe de altă parte însă, la un moment dat, îndepărtăm doar un sfert din aria rămasă, lăsând mare parte încă acoperită și, deci, oricât de mult am repeta această operațiune, întotdeauna va rămâne mai mult decât s-a luat. Deci aria nu ajunge niciodată zero.

Acest tip de problemă l-a tulburat pe *Leibnitz* și a ridicat dificultăți matematicienilor încă din antichitate. Dar, să vedem, totuși, cât este aria *triunghiului lui Siepinski.* Chiar dacă intuiția sugerează că ea ar fi zero, există o demonstrație care ne arată că ea nu poate fi zero. Cu toate acestea, comunitatea matematicienilor a hotărât ca aria să fie considerată zero, acest lucru datorându-se în principal imposibilității răspunderii la întrebarea: ***"Dacă aria nu este zero, atunci cât este?"***. Neputând combate acest lucru, *Sierpinski* și colegii lui au admis că problema ariei infinit de mici, dar diferită de zero a fost doar "un vis urât".

Termeni...

Sistem dinamic: un sistem a cărui stare la un moment dat poate fi determinată din starea lui la un moment anterior și dintr-un set de ecuații care modelează felul în care iterează sistemul.

Entropie: Proprietatea matematică a unui sistem, care îl face incapabil pe un observator să distingă între ele părțile acestuia; o măsură a factorului aleator (ΔS).

Fractal: orice formă ale cărei părți componente, atunci când sunt mărite, prezintă tot atâtea detalii ca și întregul. Unii matematicieni folosesc cuvântul fractal cu sensul de formă care este strict autosimilară și nu statistic autosimilară (potrivit acestei definiții, seria *Mandelbrot* ***nu este un fractal***). Sensul original se referă la o formă a cărei dimensiune *Haussdorff-Besicovitch* depășește dimensiunea sa topologică.

Curbe de umplere a spațiului

Chiar dacă lui *Sierpinski* i s-a impus acceptarea faptului că aria unei forme alcătuite doar din linii este zero, matematicienii au încercat să demonstreze contrariul.

O altă formă imaginată de același matematician este *covorul lui Sierpinski*. Forma geometrică de la care se pornește este un segment de dreaptă care este apoi înlocuit cu alte opt segmente de dreaptă (așa cum se poate vedea în figura 5). Fiecare segment este înlocuit cu forma întreagă și procesul continuă la infinit. Asemănându-se foarte tare cu *triunghiul lui Sierpinski* și putând fi obținut prin metoda prezentată mai sus, dar și prin decuparea de găuri într-un pătrat plin, s-ar putea ivi și aici aceeași controversă legată de aria formei.

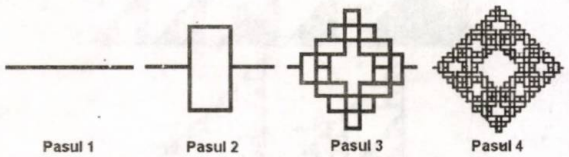


Figura 5: Covorul lui Sierpinski

Plecând de la *covorul lui Sierpinski*, matematicianul *Giusseppe Peano* a descris forma numită ***curba de umplere a spațiului a lui Peano***. Realizarea ei diferă doar prin adăugarea unui al nouălea segment de dreaptă, așa cum se poate vedea în figura 6.

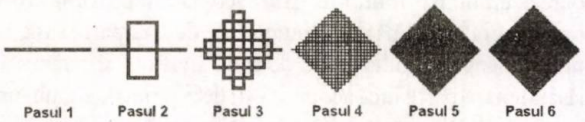


Figura 6: Curba de umplere a spațiului a lui Peano

Folosind această curbă pe care a prezentat-o în anul 1890 (când avea titlul de ***profesor extraordinar de calcul infinitezimal*** la *Universitatea din Torino*), *Peano* a demonstrat că se poate umple o porțiune din spațiu, folosind o curbă continuă care nu are lățime (deci nu are arie). În consecință, *aria curbei de umplere a spațiului* este egală cu aria în care este înscrisă. Așa s-a ajuns ca o formă alcătuită din segmente de dreaptă să umple suprafața unui plan bidimensional.

Seria Mandelbrot

Vom prezenta acum detalii referitoare la o categorie specială de fractali, și anume seria *Mandelbrot*.

Există numeroase descrieri ale acestei serii. După părerea multora, amintește de o țestoasă văzută de sus (***Randow*** - 1986). Alții văd în ea un interesant cactus "înmugurit" (***Clausberg*** - 1986).

Pentru unii matematicieni nu e mai mult decât o hipocicloidă căreia i s-au adăugat o serie de cercuri, pe care stau alte cercuri ș.a.m.d. (***Durandi*** - 1987).

De altfel, cu toții au dreptate, ca și profesorul din ***Bremen*** căruia forma îi amintește de un ***birocrat gras*** (după o rotație de 90°).

Atitudinea și interpretările personale, cu siguranță, își au rolul lor în descrierea unei imagini atât de neobișnuite. E vorba, probabil, de două feluri de percepție. În primul rând, o anumită familiaritate cu structurile asemănătoare formelor naturale, în particular cele observate la mărirea imaginii.

Pe de altă parte, ciudățenia acestui tip de repetiție este foarte diferită de cea a unei frunze sau a unui copac. Structura la diferite scări, unde apar mai multe proprietăți și detalii, contrazice viziunea noastră normală, ne face să ne revizuim percepțiile și în final să ajungem la o înțelegere. În figura 7 puteți vedea o "hartă" a *seriei Mandelbrot*.

Figura în sine nu e mult mai veche decât numele; în primăvara anului 1980, *Benoit Mandelbrot* a reușit pentru prima oară să schițeze acest grafic pe un calculator.

Niciodată n-a mai atins un produs al cercetării matematice ezoterice un asemenea nivel de popularitate, făcându-și apariția în agende, ocupând timpul liber, și devenind atât de rapid un "superstar".

Pentru a evita confuzia în descrierea ce urmează, să definim mai întâi termenii de bază:

Termeni...

Infinitezimal: un număr infinit de mic, folosit de *Leibnitz* pentru a înțelege cum funcționează calculul diferențial. S-a dovedit, în cele din urmă, că nu se ajunge la rezultate satisfăcătoare în acel mod, motiv pentru care aproape nimeni nu mai vorbește acum despre infinitezimali.

Curbă continuă: o curbă ale cărei puncte se învecinează (de fiecare parte a lor) cu alte puncte al curbei. Pentru curbele continue se poate găsi întotdeauna o linie care aproximează panta sau direcția sa în orice punct dat. Curbele fractale pot părea uneori continue, dar nu se pot determina aproximații ale pantei.

Curbă discontinuă: o curbă care conține întreruperi sau schimbări bruște de direcție; deoarece fiecare punct dintr-un fractal conține (ca întreg) toate direcțiile de mișcare, se spune că direcția se schimbă infinit de rapid și că fractalii sunt curbe infinit discontinue.

Seria Mandelbrot: Seria de puncte care nu scapă către infinit când este iterată funcția $z^2 + c$, unde c este punctul însuși și z începe din originea (0, 0). În sens mai general, orice serie de puncte care nu scapă către infinit, când este iterată o funcție care include punctul însuși drept constantă. O serie *Mandelbrot* este întotdeauna o hartă a unui număr infinit de serii *Julia* înrudite.

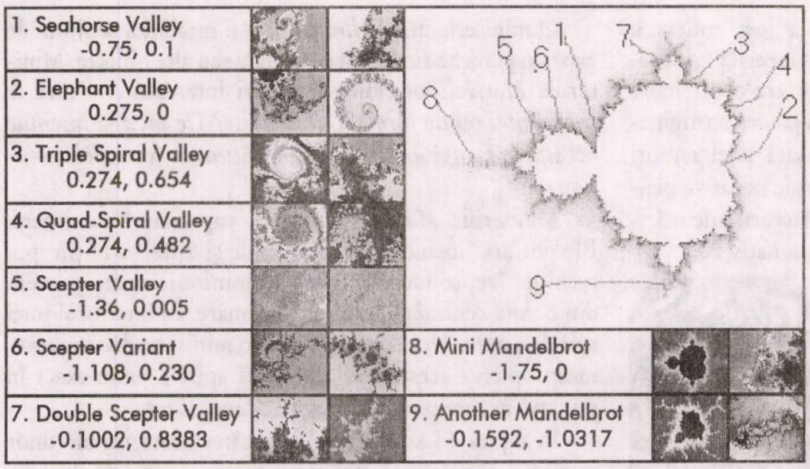


Figura 7: Harta seriei Mandelbrot

- În *seria Mandelbrot* distingem corpul principal și *mugurii*, sau *copiii Gingerbread Man.*
- La o anumită distanță de vizibil conexa regiune centrală găsim serii *Mandelbrot* de ordin superior sau sateliți, legați de corpul principal prin filamente. Singurul filament care poate fi clar observat este de-a lungul axei reale pozitive. Dar există astfel de conexiuni și către punctele aparent izolate din partea de sus și de jos a figurii.
- Vorbim de **mărire** când doar o porțiune din figură este desenată și analizată.

- Imaginea din figura 8 este numită *seria Mandelbrot* sau *Gingerbread Man*. Este bazinul *atractorului finit.*

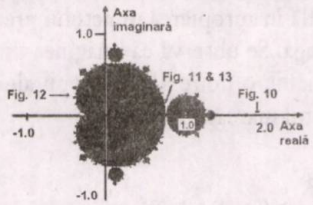


Figura 8: Seria Mandelbrot (60 de repetiții)

- Aproximațiile asupra figurii, (prezentate în figura 9) sunt liniile de contur sau suprafețele echipotențiale. Ele aparțin bazinului atractorului ∞.

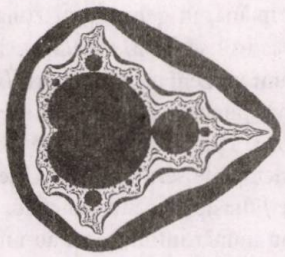


Figura 9: Seria Mandelbrot (100 de repetiții)

În figura 10 este prezentată o *serie Mandelbrot* de ordinul II.

Figura 10: O serie Mandelbrot de ordinul II

Imaginile care urmează prezintă secțiuni din varianta originală a unui *Gingerbread Man*, în diferite puncte și scalate (mărite) diferit. În figura 8, regiunile respective sunt marcate cu săgeți.

Figurile 11 și 12 prezintă o secțiune impregnată cu spirale și multe alte forme. Regiunile din partea stângă sunt excrescențe (mlădițe) ale corpului principal. La o rezoluție mai bună, ar putea fi identificate cu niște copii *Gingerbread Man.*

Figura 11: O secțiune între corpul principal și un mugure

Termeni...

Atractor: când o funcție matematică este iterată ea poate converge către o anumită valoare sau un set de valori, indiferent de valoarea inițială. Această valoare sau set de valori este denumită atractor, deoarece toate celelalte valori se deplasează către ea când funcția este iterată. Dacă o funcție are două variabile, se pot trasa numerele corespunzătoare pe un ecran bidimensional pentru a putea vedea forma atractorului. Similar, funcțiile cu trei variabile pot avea atractori tridimensionali. Deoarece majoritatea fractalilor sunt desenați prin iterarea unei funcții și trasarea valorilor către care ea converge, formele fractale pot fi considerate picturi ale atractorilor.

Iterație: procesul de aplicare repetată a unei funcții asupra rezultatelor furnizate de aceeași funcție.

Sistemul Funcției Iterate (IFS): orice funcție care folosește iterația. Mai concret, *IFS* este un sistem care itterează un set de transformări afine contractive, folosind tehnicile inițiate de *Michael Barnsley* și alți matematicieni. Aceste sisteme pot fi exprimate ca un mic set de coduri *IFS*.

Stare inițială: toate proprietățile măsurabile ale unui sistem, în momentul în care începe observarea sistemului.

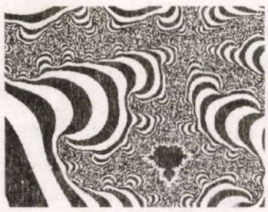


Figura 12: Un satelit foarte îndepărtat

Figura 12 a fost rotită cu 90°. În poziția normală mica serie *Mandelbrot* are o orientare aproape opusă cu cea a originalului. Este atașată unei ramuri cu valori puternic negative pentru c_{real}. Rata (factorul) de mărire este de aproximativ 500.

În figura 13 este prezentată o secțiune a fractalului aflată sub cea prezentată în figura 11.

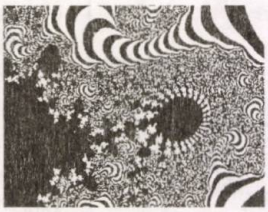


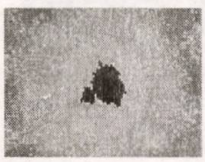
Figura 13: Regiunea de sub figura 11

Autosimilaritatea observată aici este întâlnită și în natură. Un exemplu în acest sens ar fi ramificarea (iterată) a două ramuri care pornesc din aceeași tulpină principală.

Diferența constă în faptul că autosimilaritatea fractalilor matematici nu are limite. Într-un curs, profesorul *Mandelbrot* a prezentat o imagine a unei secțiuni din *Gingerbread Man* mărită de $6 \cdot 10^{23}$ ori (numărul lui *Avogadro*), care respecta încă forma standard.

Dacă privim cu atenție, vom observa că miniseriile *Mandelbrot* (copiii *Gingerman*) sunt întâlnite, practic, peste tot sub forma unor imagini miniaturale ale seriei *Mandelbort*. Este interesant faptul că, pe lângă apariția lor în interiorul unei serii *Mandelbrot* "mai mare", îi întâlnim în cadrul unor formule neînrudite.

Miniseriile *Mandelbrot* conțin la rândul lor toată gama de caracteristici ale unei serii *Mandelbrot*, incluzând toți mugurii, căluții de mare și spiralele. Fiecare miniserie *Mandelbrot* prezintă de asemenea ***dendrite*** ale căror forme depind de particularitățile fiecărei miniserii. Aceste "dendrite" pot fi spirale, fulgere luminoase și se atașează în grupe de două, patru sau opt la miniseria *Mandelbrot* și la căluții săi de mare.



O miniserie *Mandelbrot* poate fi întâlnită acolo unde există o pereche simetrică de spirale sau alte obiecte. Miniseriile *Mandelbrot* ridică întrebări interesante: *De ce le conțin mai multe formule neînrudite? Ce vrea să însemne această prezență universală, în legătură cu formulele menționate?*

Miniseriile *Mandelbrot* au o "capacitate" uimitoare. Ele pot lua "include" împrejurimile și astfel se obțin perechi de "reproduceri". Astfel, o miniserie poate proveni din *ochiul central* al unui căluț de mare. În interiorul unei spirale a unui căluț de mare se află o miniserie *Mandelbrot*, în apropierea acesteia se află două spirale unite exact în punctul în care se află miniseria *Mandelbrot*.

În figura 14 sunt prezentate câteva imagini ale unor miniserii *Mandelbrot*. Puteți vedea opt perechi de tipul celor descrise anterior. Imaginea din stânga a fiecărei perechi prezintă o structură de tip *Mandelbrot*; imaginea din dreapta dezvăluie împrejurimile unei *miniserii Mandelbrot* care se află în apropierea punctului prezentat în imaginea din stânga. Se observă că imaginea din dreapta este, în mod invariabil, o unire a două copii ale imagii de sus, unite în punctul marcat.

Seriile Julia

Există o întreagă familie de fractali numiți serii *Julia*, care sunt îndeaproape înrudiți cu seria *Mandelbrot*. De fapt, în interiorul fiecărui punct din cadrul unei serii *Mandelbrot* se află o întreagă serie *Julia*.

Seria *Julia*, asociată cu un punct dat din cadrul seriei *Mandelbrot*, seamănă, în general, cu zona ce înconjoară acest punct în cadrul seriei *Mandelbrot*. În cadrul hărții din figura 15, sunt prezentate câteva serii *Julia* foarte interesante, precum și punctele corespunzătoare lor din cadrul seriei *Mandelbrot*. După cum se poate observa, există o mulțime de varietăți ale seriilor *Julia*. Unele modele sunt evidente. Seriile *Julia* aparținând punctelor din afara seriei se găsesc într-un număr infinit și nu au un "lac" negru în interior. Seriile *Julia*, care aparțin punctelor de la capetele dendritelor, sunt ele însele dendritice ca formă. Cele care

Termeni...

Scăparea către infinit: În cazul anumitor funcții matematice, cum ar fi cea folosită pentru desenarea seriei *Mandelbrot*, dacă un punct scapă dintr-o anumită zonă atunci când funcția este iterată, se poate demonstra că în cele din urmă ea se va deplasa rapid către infinit. De aceea, se spune că punctul scapă către infinit , imediat după ce părăsește zona

Algoritmul timpului de scăpare: o tehnică pentru afișarea hărților de funcții iterate; punctele sunt colorate în funcție de cât de repede scapă dintr-o zonă, atunci când funcția este iterată.

Serie Julia: orice serie care conține doar puncte care rămân stabile pe pacursul iterării unei funcții. Mai concret, o serie *Julia* asociată cu funcția $z = z^2 + c$, unde c este o constantă arbitrară.

Punct fix: Punctul în care se sfârșește o transformare, când este iterată. Fiecare transformare afină contractivă are un punct fix.

Cosinus: funcție trigonometrică prin a cărei iterație se obține un atractor care se bifurcă cu cicluri de toate lungimile posibile, incluzând regiuni complet haotice.

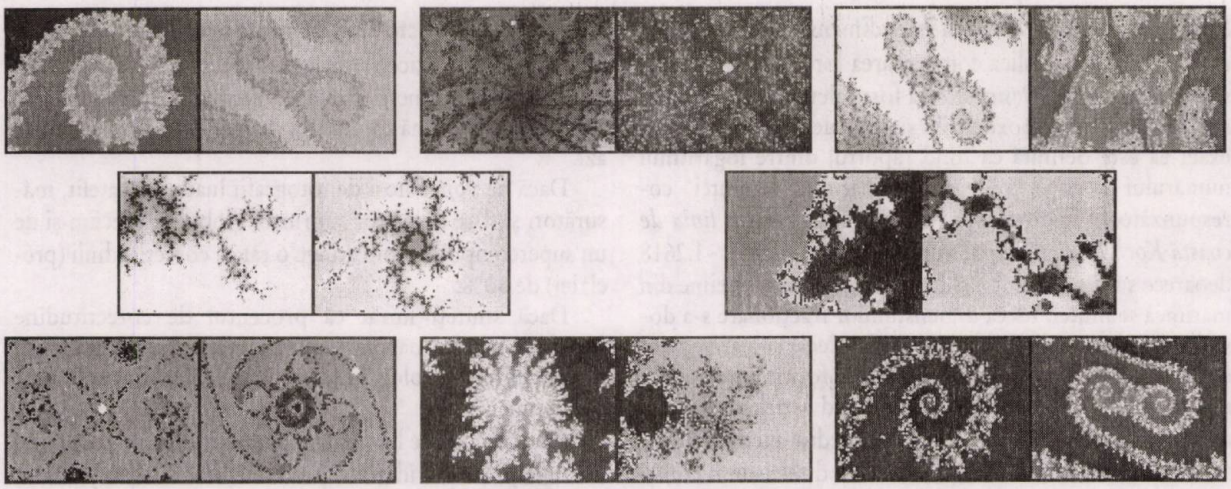


Figura 14: Exemple de miniserii Mandelbrot

aparțin seriei au zone asemănătoare cu un lac negru interior.

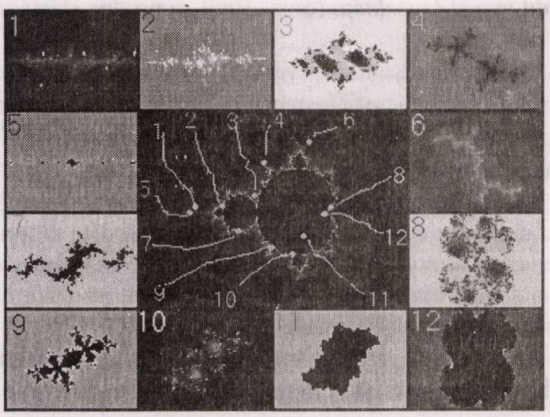


Figura 15: Harta seriilor Julia

În seriile *Julia* există modele care sunt înrudite cu miniseriile *Mandelbrot*. Aceasta se poate observa în imaginile din figura 16.

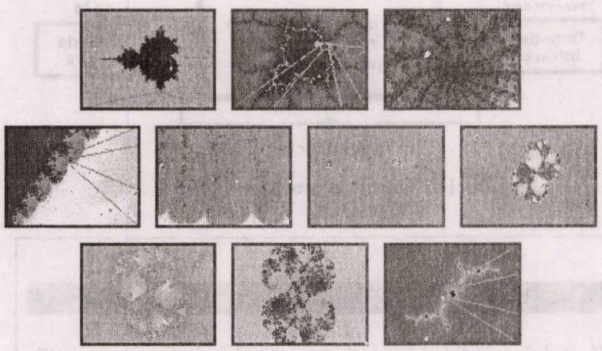


Figura 16: Modele din seriile Julia

Miniseriile *Julia* pot fi întâlnite în cadrul seriilor *Julia* ale căror puncte se află în apropierea unei miniserii *Mandelbrot*, și în seria *Mandelbrot* însăși în cadrul dendritelor unei miniserii *Mandelbrot*. În figura 17 sunt ilustrate două serii *Julia*.

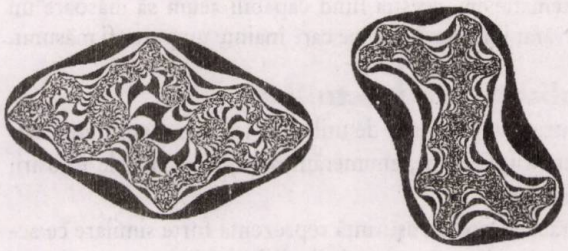


Figura 17: Serii Julia

Dimensinea fractală

Este arhicunoscut faptul că orice formă are o anumită dimensiune. Astfel:

- punctul are dimensiunea 0;
- dreapta are dimensiunea 1;
- planele și suprafețele au dimensiunea 2;
- spațiul are dimensiunea 3.

Aceste dimensiuni au fost mai apoi extinse, ajungându-se la așa-numitele spații *n*-dimensionale. În aceste spații, obiectele vor avea *n* dimensiuni. De exemplu, într-un hiperspațiu cvadridimesional (cu patru dimensiuni) vom avea hipercuburi (obiecte cvadridimensionale ale căror secțiuni sunt cuburi tridimensionale), hipersfere (obiecte cvadridimensionale ale căror secțiuni sunt cuburi tridimensionale). În acest caz, prin secțiune se înțelege intersecția obiectului cvadridimensional cu un spațiu tridimensional. În general, prin secțiunea unui obiect *n*-dimensional se înțelege intersecția acestuia cu un spațiu (*n* - 1)-dimensional.

Noile descoperiri din matematică, realizate la începutul secolului al XX-lea, au dus inevitabil la necesitatea unui mod cu totul nou de măsurare a spațiului și a dimensiunii. Răspunsul a fost dat de către matematicienii ***Felix Haussdorff*** și ***Abram S. Besicovitch*** care au descoperit (în sensul strict al cuvântului) dimensiuni noi, dar au și redefinit cu adevărat dimensiunea în sine.

În urma studierii și dezvoltării lucrărilor lui *Haussdorff*, *Besicovitch* a afirmat că formele pot avea dimensiuni fracționare. Astfel, curbe cum ar fi cele ale lui

Sierpinski sau *Koch*, ar cădea între dimensiunile normale și astfel s-ar putea explica comportarea lor ciudată. Dimensiunea *Haussdorff-Besicovitch* a fost calculată pe baza măsurătorilor de la aproximările simple ale unei curbe. Mai exact ea este definită ca fiind raportul dintre logaritmul numărului de copii și logaritmul mărimii "seminței" corespunzătoare fiecărei copii. De exemplu, pentru *linia de coastă Koch triunghiulară*, vom avea log 4 / log 3 ~1.2618 deoarece sunt patru copii și fiecare reprezintă o treime din mărimea seminței. Ideea dimensiunilor fracționare s-a dovedit până la urmă mult mai puternică decât și-au imaginat creatorii ei, deoarece natura abundă de forme auto-reflective (cum ar fi liniile de coastă) putând astfel caracteriza după acest criteriu o mare parte din mediul înconjurător.

Mai târziu, *Benoit Mandelbrot* a dat dimensiunii *Haussdorff-Besicovitch* numele de ***dimensiune fractală***. Această dimensiune fractală a avut o deosebită valoare pentru matematicieni, aceștia fiind capabili acum să măsoare un adevărat univers de forme care înainte nu putea fi măsurat.

Aplicații ale fractalilor

Pentru a vă convinge de utilitatea tot mai mare și mai evidentă a fractalilor, enumerăm mai jos avantajele folosirii acestora:

- Fractalii pot cu ușurință reprezenta forțe similare ce acționează la mai multe niveluri ale scării, lucru ce nu poate fi realizat cu geometria liniară.
- Cu ajutorul unui calculator se poate realiza ușor o curbă fractală asociată unui set de date.
- Fractalii oferă o metodă de stocare a imaginilor și datelor mult mai compactă decât vectorii liniari.
- Dacă ecuațiile liniare dau greș în construirea unor sisteme intrinseci, imprevizibile și haotice, atunci realizarea acestora este posibilă cu ajutorul fractalilor.
- Cu ajutorul fractalilor, iregularitățile devin părți esențiale ale modelului, în loc să reprezinte zgomote ce trebuie eliminate.

Toate aceste avantaje prezentate mai sus (și suntem siguri că am enumerat doar o mică parte din totalul avantajelor) au dus la adoptarea fractalilor în multe domenii, cum ar fi: meteorologie, seismologie, cardiologie, economie ș.a.m.d. De altfel, în continuare vom explica mai pe larg aplicarea teoriei fractale, a haosului și a sistemelor dinamice în cadrul meteorologiei.

Aplicații ale fractalilor în meteorologie

Oricine poate prezice vremea cu o precizie de 100% și este sigur că vom obține o rată a corectitudinii (preciziei) de 60%, dacă se afirmă că vremea de mâine va fi ca și cea de azi.

Dacă ne vom folosi de fotografii luate din satelit, măsurători și date de pe tot cuprinsul globului, precum și de un supercomputer, vom atinge o rată a corectitudinii (preciziei) de 80%.

Dacă sunteți mirat că procentul de corectitudine (precizie) nu este mai mare, atunci trebuie să priviți modul în care un meteorolog își construiește presupunerile despre vreme.

Meteorologii se bazează în predicțiile lor pe ***principiul cauzalității*** (cauze identice produc efecte identice) dar, în practică, nu se poate afla cu precizie cum va fi vremea, deoarece este imposibilă situarea unui număr foarte mare de stații meteorologice în puncte arbitrar alese pe glob (și asta în fiecare zi). De aceea meteorologii au apelat la ***principiul tare al cauzalității*** (cauze similare produc efecte similare).

În ultimele decenii s-au realizat modele teoretice care, cu ajutorul unor ecuații matematice extrem de complicate, calculează (evident cu ajutorul calculatorului) posibilele evoluții ale vremii.

Acestor sisteme li se pun la dispoziție date despre vremea de pe tot cuprinsul globului (cum ar fi date despre: presiunea atmosferică, temperatură, direcția vântului ș.a.m.d.), pe baza cărora calculează vremea rezultată cu ajutorul modelului subiacent

Un sistem de predicție meteorologică poate fi ilustrat grafic așa cum se arată în figura 18.

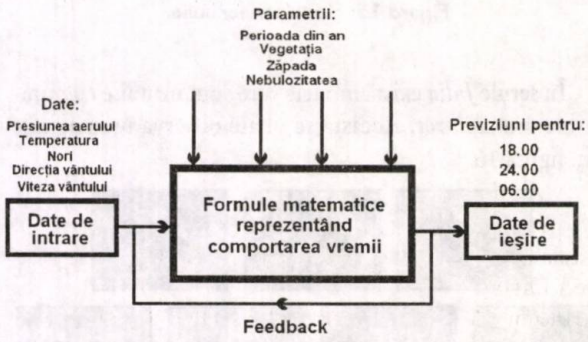


Figura 18: Sistem de predicție meteorologică

Termeni...

Dimensiune fractală: O măsură a neregularității sau asprimii unei forme; gradul în care forma "umple spațiul". Nu s-a găsit încă o definiție exactă a dimensiunii fractale, cu atât mai puțin o formulă generală pentru calcularea ei. În general, este estimată calculând raportul logaritmic al unor proprietăți la diferite scări.

Efectul fluturelui: ideea că o mică schimbare în condițiile inițiale, cum ar fi un fluture care dă din aripi de cealaltă parte a *Globului*, poate crea o variație mare în condițiile finale, cum ar fi vremea de mâine. În termeni tehnici, aceasta se numește ***dependența sensibilă de condițiile inițiale***.

Dacă privim înregistrările pe termen lung ale datelor climaterice observăm existența ciclurilor autoreflective, cum ar fi perioade mari de secetă (de exemplu înregistrările realizate pe fluviul *Nil* au relevat perioade de secetă de un mileniu, iar studiile efectuate asupra izotopilor din adâncurile măilor indică variații de temperatură la un interval de mii de ani).

Aceste cicluri autoreflective nu apar doar pe perioade mari de timp, ci și pe perioade mai mici, ca de exemplu o săptămână sau o lună (uneori chiar pe parcursul aceleiași zile).

Pentru caracterizarea acestui tip de date, tehnicile liniare nu ne oferă nici o cale ușoară de caracterizare; fenomenul pierzând orice aparență de structură cu cicluri îmbinate unul în altul (această structură fiind de fapt cel mai interesant de modelat).

De cealaltă parte avem varianta utilizării unei curbe fractale aproximate, pe care putem modela mult mai ușor caracterul de cicluri îmbinate unul în altul. Deși are mari dezavantaje la nivel punctiform (față de modelul liniar și aleator) ne sugerează mult mai exact caracterul esențial pe care îl are curba. Acest model nu este unul anticipativ, deci dacă este folosit pentru prezicerea vremii din următoarea dată de 1 decembrie (de exemplu), previziunea va fi statistic mai puțin exactă decât o previziune bazată pe mediile anuale.

Dar să vedem în continuare câteva afirmații care sunt adevărate doar în cazul datelor reale și a modelului fractal, nu și pentru modelul liniar și aleator:

- variația medie a temperaturii rămâne constantă în timp;
- variația medie a temperaturilor rămâne constantă într-o gamă foarte largă de scări;
- variația, exprimată ca și dimensiune fractală, este cuprinsă între 1.4 și 1.5;

În fiecare zi se cheltuiesc sume uriașe de bani pe echipamente computerizate care vor fi folosite pentru prezicerea vremii, cu toate acestea previziunile meteorologice sunt renumite tocmai pentru inexactitatea lor.

Nici folosirea fractalilor în jocul previziunilor meteorologice nu a adus îmbunătățiri spectaculoase, dar ei au fost de un mare ajutor explicând de ce ele nu dau rezultatele scontate.

În anul 1961, marele meteorolog ***Eduard Lorentz*** de la *M.I.T.* (*Massachusetts Institut of Technology*) a descoperit, oarecum din întâmplare, ceea ce a fost introdus în istorie sub numele de ***efectul fluturelui***. Efectul fluturelui este cunoscut mai exact ca ***dependența sensibilă de condițiile inițiale*** și este o caracteristică specifică sistemelor naturale și sociale complexe.

Fractalii în zilele noastre

Începând cu anii 1990 fractalii sunt folosiți pe scară largă, mai ales în domenii cum ar fi grafica pe calculator. Astfel, producții cinematografice importante își bazează efectele speciale pe fractali, iar sistemele grafice profesionale îi folosesc în realizarea diferitelor structuri naturale.

Dacă în trecut matematicienii și oamenii de știință "se temeau" de fractali, acum ei îi folosesc ca pe niște unelte indispensabile și de mare valoare pentru munca lor.

Prin urmare știința, matematica și tehnologia nu vor mai fi privite niciodată ca fiind domenii inestetice sau plictisitoare.

Odată cu dezvoltarea tehnicii de calcul, prin realizarea unor procesoare de interacțiuni tot mai rapide, instrumentele de desenare fractală vor deveni probabil parte integrată a tuturor sistemelor grafice pe calculator.

Bibliografie

1. Karl-Heinz Becker, Michael Dorfler, *Dynamical Systems and Fractals*, Cambridge University Press, United Kingdom, 1983.
2. *Notices of the American Mathematical Society - Computers and Mathematics*, American Mathematical Society, March 1991, Volume 38, Number 3.
3. Dick Oliver, *Fractali*, Editura Teora, 1996.
4. Jaz R. Hill, *The Mandelbrot Set*, 1997
5. Florin Munteanu, *Între ordine și dezordine: o muchie a haosului*, 1997

Ciprian Ileană este student în anul II la Universitatea Babeș-Bolyai din Cluj și poate fi contactat prin e-mail la adresa **ciprian_il@yahoo.com**. *Ioana-Maria Ileană este studentă în anul II la Universitatea Politehnica din București și poate fi contactată prin e-mail la adresa* **willow@email.ro**.

Termeni...

Jocul haosului: Denumirea favorită a lui *Michael Barnsley* pentru algoritmul iterației aleatoare. Aceasta nu este o expresie tocmai exactă, pentru că algoritmul nu prezintă o comportare haotică în sensul strict al cuvântului. *Barnsley* folosește cuvântul haos înțelegând prin aceasta factorul aleator.

Echilibru: O stare stabilă, în care cantitatea de energie care intră într-un sistem este egală cu cantitatea de energie care iese din sistem. Majoritatea sistemelor studiate în știința haosului sunt departe de echilibru.

Știința haosului: studiul sistemelor haotice.

Realizarea interfețelor CONVERSAȚIONALE

Ion-Gelu Tomșa

O interfață conversațională reprezintă o nouă formă, îmbunătățită de interacțiune cu utilizatorul. Pe lângă interfața clasică mouse-tastatură, realizatorii de programe au acum la dispoziție includerea recunoașterii vorbirii și a sintezei vocale în aplicații. Pentru vorbitorii de limbă engleză aceasta nu mai este de domeniul science-fiction, aplicațiile mai mult sau mai puțin profesionale fiind deja disponibile.

Tehnologia Microsoft Agent

În acest articol vom încerca să prezentăm o modalitate de interacțiune cu computerele, actuală și în mod cert de viitor, în contextul impactului noilor tehnologii informaționale în cele mai diverse domenii de actualitate. Facilitățile multimedia ale calculatoarelor de astăzi au atins performanțe care, până nu demult, erau considerate greu de realizat, ceea ce a dus la apariția unor interfețe tot mai ușor de folosit și tot mai apropiate de utilizator. Aplicații ca sinteza vocală și recunoașterea vorbirii au părăsit de puțin timp laboratoarele sofisticate de cercetare și sunt integrate în aplicații software utilizate zilnic.

Dacă interfața clasică de tip *point-and-click*, completată de tastatură, a sistemului de operare *Microsoft Windows* vi se pare oarecum desuetă și doriți să aduceți viitorul mai aproape de calculatoarele mileniului III atunci nu trebuie neglijată oferta *Microsoft*: un set de servicii software numit *Microsoft Agent* pentru includerea în aplicațiile *Windows* sau în paginilor *Web* a unor "asistenți" interactivi vorbăreți.

Această tehnologie, relativ nouă, ajunsă la versiunea 2.5 încearcă să aducă un nou concept de interacțiune între utilizator și calculator și constituie un punct de plecare în realizarea unor interfețe conversaționale.

Apărută în mai 1997, versiunea *Microsoft Agent 1.0* nu a fost larg distribuită, fiind urmată de prima versiune (1.5) pusă la dispoziția programatorilor. Inițial, tehnologia a oferit suport pentru limba engleză, iar de la versiunea 2.0, pe lângă alte îmbunătățiri, s-a trecut la internaționalizarea acestei tehnologii, asigurându-se suport pentru majoritatea limbilor de circulație internațională și posibilitatea de extindere pe viitor a serviciilor oferite, existând inclusiv un suport limitat și pentru limba română.

Interfața software *Microsoft Agent API* furnizează un pachet complex de servicii software pentru afișarea și animația unor personaje care reacționează la acțiunile utilizatorului. Răspunsul personajului este fie afișat într-un balon de tipul benzilor desenate, fie exprimat prin sinteza vocală.

Implementată ca un server *COM* (*Common Object Model*) *OLE Automation*, această interfață software permite mai multor aplicații, numite clienți sau aplicații-client, să găzduiască și să aibă acces în același timp la serviciile de animație, recunoaștere și sinteză vocală. Un client poate fi orice aplicație care se conectează la interfața *COM* a *Microsoft Agent*.

Ca server *COM*, *Microsoft Agent* este încărcat automat numai când un client apelează interfața *COM* și cere conectarea la aceasta. Serverul *COM* rulează până când toți clienții își închid legăturile, iar în momentul în care nu mai sunt clienți conectați, *Microsoft Agent* se închide automat. Deși interfața *COM* poate fi apelată direct, este inclus și un control *ActiveX*, care asigură un acces facil la serviciile *Microsoft Agent* din limbaje de programare care oferă o interfață cu acest tip de controale.

În plus, pe lângă programele *Windows*, tehnologia *Microsoft Agent* poate fi inclusă prin *script*-uri în pagini *Web*, care pot fi rulate cu browsere care includ suport pentru *ActiveX*. Browserul *Microsoft Internet Explorer* oferă suport pentru *ActiveX* ca și pentru limbajele de *script* cu care poate fi programat agentul, cum ar fi *VBScript* sau *JScript* / *JavaScript*.

Microsoft Agent nu își propune să înlocuiască interfața clasică (mouse-tastatură), ci este o extensie a interfeței *Microsoft Windows*. Pentru includerea personajelor animate în aplicații, *Microsoft* oferă o documentație detaliată (aproximativ 7 MB de documente *Word*), cât și o adresă *Web* (**http://agent.microsoft.com**), dedicată programatorilor și designerilor de caractere.

Accesarea controlului din pagini Web

Pentru apelarea serviciilor *Microsoft Agent* dintr-o pagină *Web*, se folosește marcajul *HTML 4.0* `<OBJECT>` inclus în corpul marcajului `<HEAD>` sau `<BODY>` al paginii, specificând identificatorul clasei `clsid` al controlului, furnizat de *Microsoft*. În plus, parametrul `codebase` poate preciza *URL*-ul programului care va instala *Microsoft Agent* și numărul versiunii. Dacă sistemul client are instalat *Internet Explorer* (versiunea 3.02 sau mai nouă), dar *Microsoft Agent* nu este încă instalat și utilizatorul accesează o pagină *Web* care include marcajul `<OBJECT>` cu parametrul `clsid` al *Microsoft Agent*, browserul va încerca să descarce automat *Microsoft Agent* de pe *site*-ul *Microsoft*. După aceea utilizatorul va fi întrebat dacă dorește continuarea instalării.

Următorul exemplu ilustrează modul de utilizare al parametrului `codebase` pentru a descărca automat versiunea 2.0 în engleză a *Microsoft Agent*. Pentru informații despre adresa actuală a fișierului de instalare *Microsoft Agent* în altă limbă și numărul versiunii disponibile, consultați **http://msdn.micosoft.com/msagent/agentdevdl.asp**.

```
<OBJECT
  classid="clsid:D45FD31B-5C6E-11D1-9EC1-
           00C04FD7081F"
  codebase="#VERSION=2,0,0,0"
  id=Agent>
</OBJECT>
```

Este posibilă instalarea interfeței *Agent* de pe propriul server sau includerea sa în programul de instalare al aplicației dumneavoastră. Pentru a permite instalarea automată de pe serverul http propriu, trebuie să copiați pe server fișierul `cabinet.exe` cu auto-instalare și să precizați adresa *URL* în parametrul `codebase`:

```
<OBJECT
  classid="clsid:D45FD31B-5C6E-11D1-9EC1-
           00C04FD7081F"
  codebase="yourURL#VERSION=2,0,0,0"
  id=Agent>
</OBJECT>
```

Pentru a obține fișierul *cabinet* de instalare automată de pe *site*-ul *Microsoft*, descărcați *Microsoft Agent* de pe site și alegeți opțiunea *Save* din cutia de dialog afișată de programul de navigare pentru a salva fișierul în directorul pe care îl alegeți.

Pentru a instala versiuni în alte limbi ale interfeței *Agent*, corpul marcajului `<OBJECT>` va fi diferit și trebuie să conțină identificatorul de clasă al componentei dorite. Instalarea componentelor pentru suportul unor limbi diferite de limba engleză poate necesita repornirea calculatorului de către utilizator. Lista componentelor disponibile pentru alte limbi poate fi obținută de pe *site*-ul *Agent*, folosind aceeași procedură ca la instalarea componentei nucleu a interfeței *Agent*. Pentru a include un personaj într-o aplicație sau pagină de *Web* proprie, acesta trebuie mai întâi încărcat prin metoda *Load* a controlului. Fișierul cu definiția personajului poate fi stocat local sau pe un server. După încărcarea personajului pot fi apelate toate metodele, proprietățile și evenimentele oferite de controlul *Agent* pentru programarea personajului. De asemenea, pot fi utilizate facilitățile unui limbaj de programare pentru a implementa reacțiile personajului la evenimente cum ar fi apăsarea butonului *mouse*-ului.

Programarea în pagini *Web* a controlului poate fi realizată folosind *VBScript* sau *JScript/JavaScript* prin marcajul *HTML* `<SCRIPT>`. Pentru accesul la interfața controlului trebuie folosit parametrul `id`, care reprezintă numele asociat controlului în marcajul `<OBJECT>`. Acesta este urmat de sub-obiect (dacă există), numele metodei sau al proprietății și orice alți parametri sau valori suportate de metoda sau proprietatea respectivă:

```
agent[.object].Method parameter,[parameter]
agent[.object].Property = value
```

Pentru evenimente, trebuie inclus numele controlului urmat de numele evenimentului și eventualii parametri:

```
Sub agent_event (ByVal parameter
                 [,ByVal parameter])
  instrucțiuni...
End Sub
```

Specificarea unui *handler* pentru eveniment poate fi realizată în corpul marcajului `<SCRIPT>`, folosind construcția `For...Event`:

```
<SCRIPT language=VBScript for=agent
  event=event[(parameter[,parameter])]>
  instrucțiuni...
</SCRIPT>
```

Dacă utilizați *JScript/JavaScript*, pentru accesul la interfața de programare, trebuie folosite convențiile acestui limbaj pentru specificarea metodelor sau proprietăților:

```
agent.object.Method (parameter)
agent.object.Property = value
```

JavaScript nu oferă evenimente pentru obiecte *non-HTML*. Totuși, pentru programul de navigare *Internet*

Explorer poate fi folosită construcția `for...event` în interiorul corpului marcajului `<SCRIPT>`:

```
<SCRIPT language="JScript" for="object"
  event="event()">
  instrucțiuni...
</SCRIPT>
```

Datorită faptului că această sintaxă este acceptată numai de *Internet Explorer*, paginile create nu vor putea fi vizualizate cu alte browsere.

Pentru a accesa colecția de obiecte ale *Agent*-ului, trebuie folosită funcția *JScript Enumerator*. Versiunile *JScript* apărute înainte de *Internet Explorer 4.0* nu recunosc această funcție și nu suportă colecțiile.

Vom prezenta un exemplu în care sunt folosite două personaje animate care prezintă o "scenetă". Comenzile din documentul *HTML* sunt, de fapt, indicații de regie, iar în textul sursă sunt inserate comentarii. Pentru obținerea unei succesiuni fluente de animații, personajele trebuie să aștepte completarea ultimei operații a celuilalt caracter prin apelul metodei `Wait`. Evident, pentru a vizualiza scena este necesară instalarea interfeței *Microsoft Agent*; de asemenea verifcați dacă aveți instalate fișierele `*.acs` cu personajele necesare în directorul `C:\WINDOWS\MSAGENT\CHARS`.

Documentul *HTML* este disponibil pentru download la **www.ginfo.ro/revista/11_7**.

Accesarea controlului din Visual Basic

...și alte limbaje de programare

Instalarea controlului Agent

Controlul *Agent* poate fi inclus în programele scrise în *Visual Basic* și alte limbaje de programare (*Visual C++*, *Delphi* etc.), care oferă interfața cu controalele *ActiveX*, prin respectarea convențiilor de acces la aceste controale. Folosirea acestui control implică instalarea în prealabil a pachetului software *Microsoft Agent* și adăugarea controlului în paleta de componente a mediului de programare utilizat. Pentru instalarea controlului folosiți opțiunile oferite de mediul integrat pe care îl utilizați.

Crearea unei instanțe a controlului Agent

Accesul la serviciile controlului *Microsoft Agent* necesită crearea, pentru început, a unei instanțe a acestuia. Cea mai simplă modalitate este de a plasa o instanță a controlului pe un *form* în etapa de proiectare a aplicației, precum și stabilirea valorii unor proprietăți ale acestuia direct în fereastra cu proprietățile controlului.

Metode, proprietăți, evenimente

Folosirea controlului *Microsoft Agent* în *Visual Basic* este similară cu folosirea acestuia în *VBScript*, cu excepția faptului că evenimentele în *Visual Basic* trebuie să includă tipul datei trimise ca parametru. Adăugarea controlului *Microsoft Agent* va include automat evenimentele controlului cu parametrii necesari. De asemenea, va crea automat o conexiune la serverul *Agent* când aplicația este executată.

În continuare este prezentat un program *Visual Basic*, în care vrăjitorul *Merlin* încearcă să vă îndeplinească dorințele, apelând la câteva pase "magice". Pentru aceasta creați un proiect nou și așezați pe un *form* o instanță a controlului *Agent* (`Agent1`) și un buton de apăsare (`Command1`). Prin *dublu-click* pe *form* și pe buton completați evenimentele `Form_Load()` și `Command1_Click()`:

```
Private Sub Form_Load() 'incarcare caracter
   Agent1.Characters.Load "Merlin",
        "c:\windows\msagent\chars\Merlin.acs"
End Sub

Private Sub Command1_Click()
  Set Assistant = Agent1.Characters("Merlin")
  Assistant.Show
  Assistant.MoveTo 250, 200
  Assistant.Speak "Hello! I am Merlin."
  Assistant.Speak "What's your wish, master?"
  Assistant.Speak
    "Ok! Your wish is granted!|That's easy!|"
  Assistant.Play "DoMagic1"
  Assistant.Play "DoMagic2"
  Assistant.Speak "Ok!|Done!"
  Assistant.Speak
    "Bye now!|Have a nice day!|See you soon!"
  Assistant.Play "Greet"
  Assistant.Hide
End Sub
```

Microsoft Agent permite construirea unor variații ale aparițiilor personajului animat. Astfel, în procedura `Speak` pot fi scrise mai multe variante ale textului corespunzător unei situații, care vor fi selectate aleator în momentul exprimării lor. Variantele dorite vor fi separate prin caracterul `"|"`. Apariția asistentului interactiv poate fi asociată unor evenimente clasice, de exemplu apăsarea unui buton sau îndeplinirea unor condiții logice.

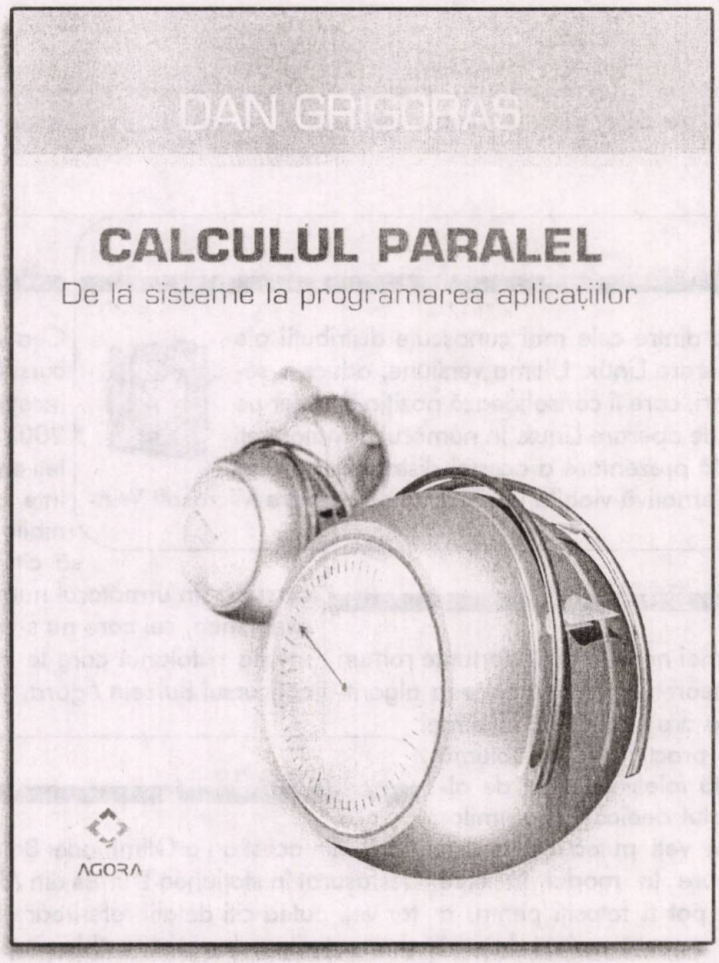
Includerea în aplicații a tehnologiei *Agent* trebuie să urmărească câteva principii de design al interfeței cu utilizatorul, altfel riscă să producă o reacție de respingere din partea acestuia. Apariția automată, și anunțarea acelorași mesaje de către asistenții interactivi nu este o opțiune foarte inspirată.

În plus, aceștia, pentru a fi înzestrați cu o personalitate proprie, trebuie să alterneze animația cu prezentarea textului. Acești "asistenți interactivi" trebuie priviți ca o îmbunătățire a interfeței clasice și nu ca o înlocuire definitivă a acesteia.

Bibliografie

- Documentația electronică *Microsoft Agent*, *Microsoft Web*, **http://agent.microsoft.com**.

COMPUTER LIBRIS AGORA



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

În numărul URMĂTOR

La mijlocul lunii decembrie va apărea următorul număr al revistei noastre. Vă prezentăm pe scurt câteva dintre articolele pe care le veți putea citi...

RedHat Linux 7.2

RedHat este una dintre cele mai cunoscute distribuții ale sistemului de operare Linux. Ultima versiune, aduce o serie de îmbunătățiri, care îi consolidează poziția de lider pe piața sistemelor de operare Linux. În numărul următor veți putea citi o scurtă prezentare a acestei distribuții, care se dorește a fi o alternativă viabilă la sistemul de operare Microsoft Windows.

Bursele Agora

Cea de-a treia etapă a concursului organizat de GInfo va începe pe data de 1 ianuarie 2002. Dacă doriți să cunoașteți enunțurile problemelor înainte ca acestea să fie disponibile online, vă recomandăm să citiți paginile dedicate concursului din următorul număr al revistei noastre. De asemenea, cei care nu s-au înscris încă, vor găsi în revistă și talonul care le dă dreptul să participe la concursul Bursele Agora.

Algoritmi genetici

Una dintre cele mai noi și mai importante ramuri ale informaticii teoretice este cercetarea algoritmilor genetici. Ea are aplicații în cele mai variate domenii, practic orice problemă poate fi rezolvată folosind astfel de algoritmi. În articolul dedicat algoritmilor evolutivi veți putea citi detalii referitoare la modul în care aceștia pot fi folosiți pentru a rezolva a gamă variată de probleme.

BOI 2001

Ediția din acest an a Olimpiadei Balcanice de Informatică s-a desfășurat în stațiunea Durres din Albania. În numărul următor veți putea citi detalii referitoare la acest concurs, precum și enunțurile celor șase probleme propuse spre rezolvare. A fost pentru prima dată când toți cei patru concurenți români au obținut medalii de aur la un concurs internațional de programare. Mai mult, la această Balcaniadă, organizatorii au fost nevoiți să suplimenteze numărul medaliilor de aur acordate (de la trei la patru), deoarece punctajele elevilor români au fost foarte mari și foarte apropiate.

O astfel de performanță nu a mai fost realizată de nici o altă țară în istoria de nouă ani a acestui concurs.

Expresii regulate

Pentru cei interesați de teoria compilatoarelor vom prezenta modul de realizare a unui parser de expresii regulate. Veți putea afla modul în care compilatoarele recunosc validitatea diferitelor elemente de limbaj cum ar fi cuvintele rezervate, identificatorii, valorile numerice etc. Doritorii vor putea să descarce fișierele sursă ale programului de pe site-ul revistei noastre.

FreePascal

FreePascal a devenit compilator oficial la Olimpiada Internațională de Informatică și la Olimpiada Europei Centrale, înlocuind compilatorul oferit de firma Borland. În numărul următor vor fi prezentate principalele diferențe dintre cele două compilatoare.

NET REPORT

PC REPORT ÎN MILENIUL III

www.pcreport.ro

Conectat la viitorul digital

AGORA MEDIA Str. Tudor Vladimirescu nr. 63/9, Tg. Mureş, CP 230/1, 4300

Tel: 065-166 516, Fax: 065-166 290, e-mail: office@agora.ro

AGORA MEDIA

OFERTĂ SPECIALĂ!

Pentru abonamentele încheiate la revista

PC Magazine România

în perioada 22 octombrie–31 decembrie

beneficiați de o

REDUCERE DE 52% !

Preț vânzare 12 numere ~~414.000~~

Abonament 12 numere* **200.000 lei**

www.pcmagazine.ro

* Cei care sunt abonați deja, pot folosi oferta pentru a-și prelungi abonamentul

Ghidul utilizatorilor de tehnologii informatice și comunicații

AGORA MEDIA S.R.L. str. C-tin Rădulescu Motru nr. 13, etaj 4, apt. 42
sect. 4 (CP 94/49), București, Tel: 01-3309282, Fax: 01-3309285

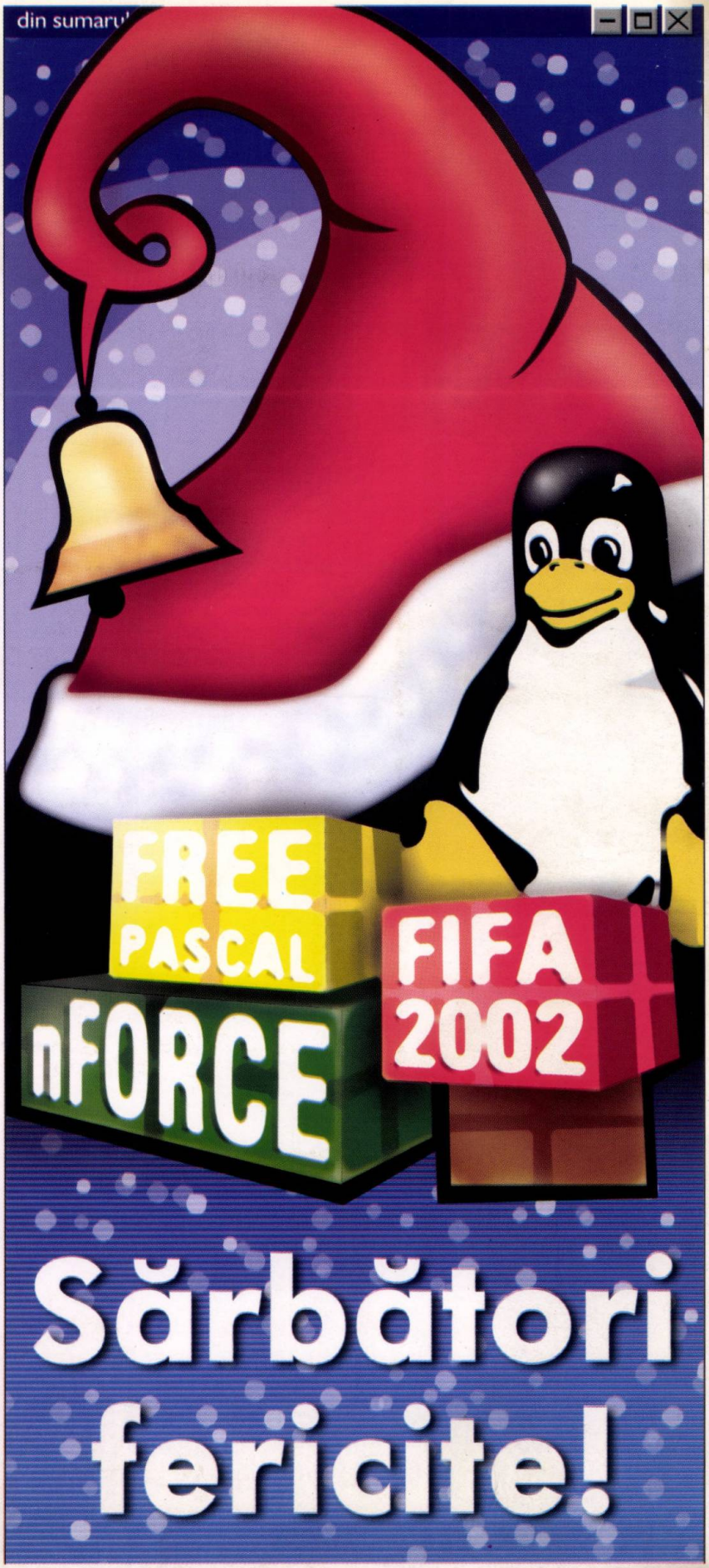
gazeta de informatica

Revista viitorilor specialiști în tehnologia informației

Decembrie 2001
Vol. 11/8

Lei 20.000

din sumar



Sărbători fericite!

AGORA
pe web
www.agora.ro
www.pcmagazine.ro
www.netreport.ro
www.eweek.ro
www.pcconcrete.ro
news.agora.ro.
www.ginfo.ro
www.aol.ro
conectează-te
ACUM!

Vol. 11, Nr. 8 /2001

gazeta de informatică

Redacția:

Director științific:
Prof. dr. *Horia Georgescu (București)*
Redactor șef:
Clara Ionescu (Cluj)
Redactor șef adjunct:
Mihai Scorțaru (Cluj)
Redactori:
Carmen Șușman (Alba Iulia)
Claudiu Soroiu (Târgu-Jiu)
Au mai colaborat:
Crina Groșan (Cluj)
Mihai Oltean (Cugir)
Mihai Stroe (București)
Anca Vasilescu (Brașov)
Oana Georgescu (Brașov)
Alexandru Șușu (București)
Tehnoredactare:
Claudiu Soroiu
Coperta și grafica:
Andrei Pașa

Adresa pentru corespondență:
Computer Libris Agora,
Universității 8/8,
3400, Cluj-Napoca
Adresa e-mail: office@ginfo.ro
Tel: 064/192.422
065/166.516
Fax: 065/166.290
http://www.ginfo.ro
ISSN: 1222-7129

Tiparul: *Master Druck* Târgu Mureș
tel: 065/162.359
Editor: Agora Media SRL
Editor al publicațiilor "eWeek România", "PC Magazine România", "NetReport" și "Gazeta de Informatică"
str. T. Vladimirescu nr. 63/9
4300 Târgu-Mureș

Director general:
Romulus Maier
Director executiv:
Adrian Pop
Director tehnic:
Szabó László
Vânzare reclamă:
Mihai Bucuroiu
Bogdan Manu
Florin Tită
Denis Marioara
Marius Comărniceanu
Reprezentanța București:
Director: *Dana Pastramă*
Tel: 01/330.92.81
Fax: 01/330.92.85

Σ Sumar

actual.. PAG. 5

Produsele lunii
nForce
RedHat Linux 7.2
FIFA 2002
MOȘ Crăciun

focus.. PAG. 30

Algoritmi EVOLUTIVI

Mihai OLTEAN
Crina GROȘAN

~prezentare generală~

probleme.. PAG. 9

propuse pentru REZOLVARE

Olimpiada Balcanică de Informatică

~Durres, Albania~

serial.. PAG. 38

Introducere în problematica sistemelor NESECVENȚIALE

Anca VASILESCU
Oana GEORGESCU

~calcul paralel~

babel.. PAG. 14

Un PARSER de expresii REGULATE

Alexandru ȘUȘU

~teoria compilatoarelor~

profil.. PAG. 45

FREEPASCAL

Claudiu SOROIU

~prezentare comparativă~

concurs.. PAG. 22

Informații generale despre concurs

Talon de participare

Problemele rundei a treia a ediției 2001/2002

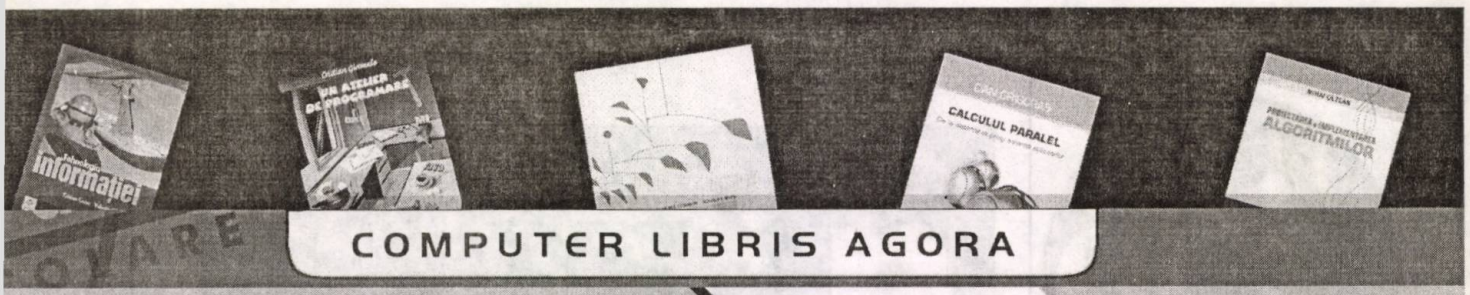
enjoy.. PAG. 46

Produse SOFTWARE

Mihai SCORȚARU

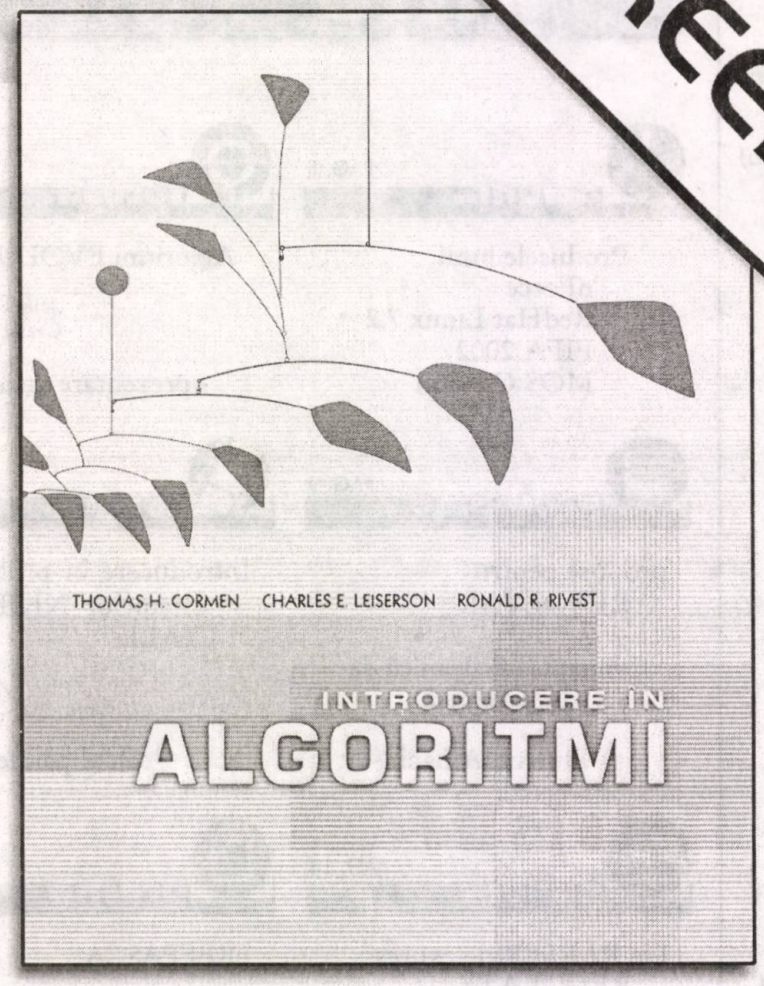
Administratori de SISTEM

Carmen ȘUȘMAN



COMPUTER LIBRIS AGORA

REEDITARE



Cartea se adresează, în primul rând, celor care studiază la nivel elementar sau academic algoritmi și structuri de date. Deoarece tratează aspecte teoretice și practice, legate de proiectarea algoritmilor, este la fel de binevenită elevilor, studenților, profesorilor și autodidacților.
Introducere în algoritmi acoperă atât metodele clasice cât și metodele moderne ca analiza amortizată și algoritmii paraleli.
Prezentarea matematică, riguroasă, este atent detaliată în așa fel încât să fie accesibilă tuturor cititorilor.
Capitolele încep cu elemente de bază, avansând spre subiecte mai complexe.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Necesitate sau banalitate...

Avem nevoie de ALGORITMI?

Mulți dintre cei care doresc să se angajeze la o firmă de software consideră că este suficient să cunoască cele mai noi limbaje de programare și cele mai noi tehnologii software apărute. De obicei, se ignoră faptul că în scrierea programelor este nevoie de cunoașterea numeroșilor algoritmi care trebuie folosiți în diferite situații.

Elevii și studenții afirmă frecvent că ceea ce învață la școală sau facultate nu le va folosi în viitor. Tinerii informaticieni în devenire susțin că limbajul *Pascal*, pe care îl studiază, nu este folosit în producție și însușirea cunoștințelor necesare pentru utilizarea sa este o pierdere de vreme. Nu cred că este cazul să polemizăm cu ei în favoarea acestui limbaj, care a fost creat în scopuri didactice și, practic, nu a fost folosit în producție. Tinerii noștri prieteni observă că, pentru a crea o interfață grafică, sunt suficiente doar câteva *click*-uri dacă se folosește un mediu vizual și nici nu vor să se gândească la efortul pe care l-ar fi depus dacă ar fi dorit să creeze aceeași interfață în limbajul *Pascal*. Nu pot decât să le dau dreptate; crearea unei astfel de interfețe grafice necesită un efort considerabil dacă nu se folosesc medii vizuale.

Se pune, totuși, problema motivului pentru care programa de liceu include studierea acestui limbaj. Explicația este foarte simplă: în liceu se urmărește însușirea unor cunoștințe de algoritmică, și nu a unor cunoștințe referitoare la ultimele tehnologii *software*.

Chiar dacă mă alătur celor care cred că aceste cunoștințe de algoritmică pot fi dobândite și în condițiile în care implementările se realizează în C/C++, cred că algoritmii sunt ***mult mai ușor de prezentat*** folosind un limbaj cu o structură cât mai simplă.

Să vedem acum ce se întâmplă atunci când elevul, studentul sau absolventul începe să lucreze la primul proiect. De obicei, creează interfața, adaugă câteva linii de cod și aplicația este gata. Nu este nevoie de nici un algoritm, ci doar de înțelegerea modului în care funcționează mediul vizual. Am întrebat un prieten care lucrează la astfel de proiecte ce se întâmplă dacă, totuși, are nevoie de un algoritm mai complicat pe care nu știe să îl implementeze, deoarece nu are cunoștințele de algoritmică necesare. Răspunsul a venit foarte repede: "*Caut pe Internet și găsesc o mulțime de implementări ale algoritmului. Le descarc și apoi le inserez în codul sursă; nu mă interesează cum au fost realizate.*"

M-am mirat puțin și apoi l-am întrebat dacă găsește chiar orice. Mi-a răspuns: "*Aproape! Am găsit tot ce am avut nevoie până acum.*"

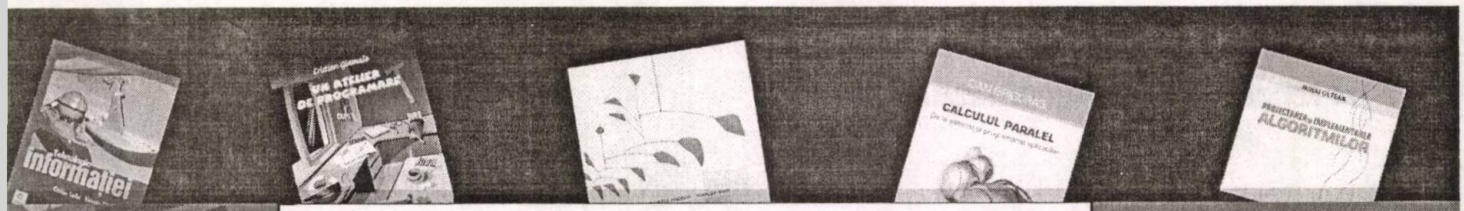
Totuși, acel "aproape" mi se pare destul de îngrijorător. Este foarte probabil ca, la un moment dat, să nu mai găsească ce are nevoie. Probabil că nu se va întâmpla așa ceva prea curând, dar cred că o astfel de situație este inevitabilă atunci când se lucrează la proiecte de anvergură.

Mai apare o problemă; datorită lipsei cunoștințelor de algoritmică este foarte probabil ca, la un moment dat, programatorul să nu își dea seama care este algoritmul de care are nevoie. Am asistat la o discuție în care un astfel de programator nu putea să își închipuie că soluția problemei sale este reprezentată de o parcurgere în lățime a unui graf. Știa că poate găsi algoritmul pe *Internet*, dar nu și-a dat seama că acesta este algoritmul de care are nevoie.

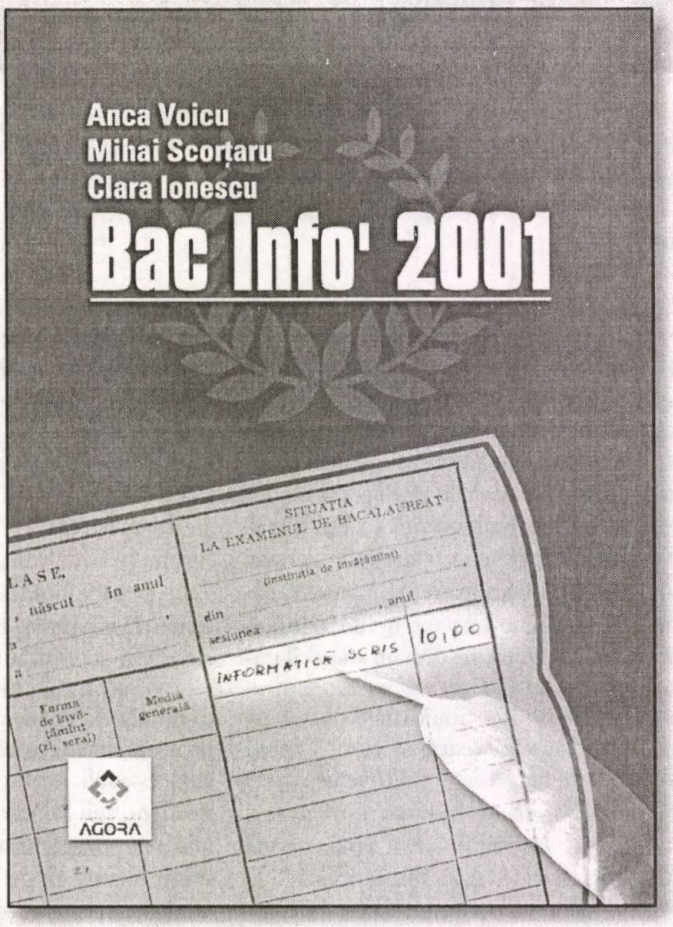
În plus, uneori acești programatori nu își dau seama că anumiți algoritmi pe care îi utilizează nu sunt eficienți. Există cazuri în care se folosesc algoritmi cu ordine de complexitate de câteva ori mai mari decât cei eficienți. De exemplu, la un moment dat, un programator se mira de ce aplicația sa nu este suficient de rapidă. Unul dintre algoritmii folosiți era cel de determinare a poziției primei apariții a unui subșir într-un alt șir. El a scris repede un algoritm care rezolva problema, dar care rula în timp pătratic. I-am spus că există *KMP* și, după ce a descărcat de pe *Internet* implementarea acestui algoritm și a inclus-o în programul său, a observat o îmbunătățire simțitoare a vitezei.

Pentru ca un programator să își desfășoare activitatea în condiții bune. sunt necesare cunoștințe de algoritmică. De ce să depinzi de *Internet*? De ce să depinzi de fructul gândirii altora? De ce să renunți la bucuria cunoașterii algoritmilor și, mai ales, la bucuria pe care o simți după o optimizare reușită?

Mihai Scorțaru



COMPUTER LIBRIS AGORA



Bac Info' 2001 conține 20 de variante de subiecte pentru Bacalaureat. Fiecare variantă conține câte patru subiecte. Primul dintre ele conține trei subpuncte care testează capacitățile candidaților de a recunoaște valoarea de adevăr a unei propoziții, de a determina efectul unui anumit algoritm și de a găsi o anumită greșeală într-un algoritm. Al doilea subiect conține doua subpuncte; primul testează deprinderea candidaților de a folosi limbajul pseudocod, iar al doilea cea de a folosi un limbaj de programare (Pascal sauC++). Al treilea subiect este o problemă pentru rezolvarea căreia este necesară folosirea unor structuri dinamice de date, iar al patrulea este o problemă pentru a cărei rezolvare trebuie folosită metoda backtracking.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

nForce

Tehnologia de realizare a plăcilor grafice s-a schimbat mult în ultimul timp. Una dintre companiile fruntașe în acest domeniu este nVidia. Cea mai nouă tehnologie dezvoltată de această firmă a fost implementată în produsul nForce, un chipset care are integrat un procesor grafic performant.

Istorie

Compania *nVidia* s-a impus pe piață, ca lider în domeniul *chip*-urilor pentru plăcile video, după lansarea produsului *GeForce* 256. Au urmat, firesc, versiunile următoare care s-au numit *GeForce* 2 și *GeForce* 3. Se pare că, în curând, va apărea pe piață și *chip*-ul *GeForce* 4.

Plăcile video integrate în plăcile de bază au apărut de câțiva ani, dar performanțele acestora nu puteau fi comparate cu cele ale celorlalte plăci video. Speranțele au apărut atunci când *nVidia* a anunțat intenția de a crea *nForce*, un *chipset* pentru plăci de bază, care să conțină o placă video integrată ale cărei performanțe să fie comparabile cu cele ale unei plăci video care folosește *chip*-ul *GeForce 2 MX* (varianta ieftină a *chip*-urilor din această familie).

Produsul a fost așteptat mult timp și, până la urmă, au apărut pe piață primele calculatoare care folosesc *nForce*. Din nefericire, tehnologia a avansat foarte repede în domeniul plăcilor video, iar perfomanțele noului *chipset* nu mai impresionează atât de mult ca în momentul anunțării intenției de a lansa produsul.

Mai mult, *nVidia* nu mai este lider autoritar pe piața plăcilor video deoarece plăcile din familia *ATI Radeon* depășesc în multe situații plăcile din familia *GeForce*.

Majoritatea specialiștilor consideră că *nForce* ar fi fost un produs excepțional dacă ar fi fost lansat cu câteva luni mai devreme.

Facilități

Majoritatea utilizatorilor, folosesc calculatoarele, în special, pentru aplicații grafice, pentru sunet și pentru comunicare în rețea.

De obicei, pentru aceasta se folosește o placă video, una audio și una de rețea. Toate acestea pot fi integrate pe placă de bază; integrarea acestora elimină necesitatea prezenței a trei plăci suplimentare în calculator.

Chipset-ul *nForce* oferă suport pentru procesoarele *Duron*, *Athlon* și *Athlon XP*, produse de *AMD* și se adresează tuturor categoriilor de utilizatori.

Produsul *nForce* are integrat un procesor pentru grafică 3D și un procesor media pentru aplicații media și comunicare în rețea.

Performanțe

Chipset-ul *nForce* oferă cel mai performant accelerator grafic integrat de pe piață. Din nefericire, performanțele acestuia sunt net inferioare majorității plăcilor video "clasice" disponibile în acest moment.

Totuși, cei care nu sunt mulțumiți de performanțele grafice ale plăcii video integrate, au posibilitatea de a insera o altă placă video. *nVidia* recomandă, în acest caz, atașarea unei plăci grafice *GeForce* 3. Apare în mod firesc întrebarea: *Ce rost are să achiziționăm o placă de bază cu facilități video integrate dacă ni se spune că ar fi bine să folosim o placă video suplimentară?*

Din păcate pentru *nVidia*, răspunsul la această întrebare este: ***nu are rost.***

Concluzii

Compania *nVidia* a realizat o nouă premieră prin introducerea *chipset*-ului *nForce*. Din nefericire, se pare că nu există nici un segment de piață căruia acesta i se poate adresa.

Pe piața calculatoarelor performante sunt preferate plăcile video *GeForce* 3 sau *Radeon* 8500 și sunt folosite plăci de bază fără facilități video integrate, dar care oferă toate celelalte facilități la același nivel al performanțelor.

Pe piața calculatoarelor ieftine, *nForce* nu poate pătrunde datorită prețului excesiv de mare. O placă de bază care folosește *nForce* costă în jur de 200$. În această situație, sunt preferate plăci de bază mai ieftine și plăci video *GeForce* 2 sau *Radeon* 7500.

Se poate spune că *nVidia* a deschis calea plăcilor de bază cu performanțe grafice ridicate. Chiar dacă actualul *nForce* lasă de dorit, ne așteptăm să apară plăci video intregrate pe placa de bază ale căror performanțe să fie echivalente cu cele ale unei plăci *GeForce* 3 actuale, sau cu ale unei viitoare plăci *GeForce* 4.

RedHat Linux 7.2

RedHat este una dintre cele mai cunoscute distribuții Linux. Recent, compania Red Hat a lansat o nouă versiune a sistemului său de operare. Versiunea 7.2 aduce o serie de îmbunătățiri, motiv pentru care RedHat Linux 7.2 a devenit produsul lunii decembrie.

Prezentare

Nu de foarte mult timp a apărut o nouă versiune a sistemului de operare *RedHat Linux*. Versiunea 7.2 este disponibilă în două variante: *Standard* și *Professional*.

După cum ne-am obișnuit cu diferitele variante ale sistemului de operare *Linux*, nu lipsește documentația care, pentru pachetul *RedHat Professional*, este acum prezentă sub forma a trei manuale.

Primul dintre acestea este un ghid care vă ajută să instalați corect și fără mari dificultăți sistemul livrat de compania *Red Hat*. Al doilea este un ghid de configurare al sistemului de operare, iar cel de-al treilea este un manual de utilizare.

Informațiile prezente în cele trei manuale se adresează categoriilor de utilizatori începători și medii. Pentru avansați aceste manuale nu aduc foarte multe noutăți.

Noutăți

În premieră, această versiune a sistemului de operare *RedHat Linux* introduce un nou sistem de fișiere, și anume ***ext3***.

În cadrul celor două pachete sunt prezente ultimele versiuni ale interfețelor grafice utilizator *GNOME* și *KDE*.

Este prezent și un nou manager de fișiere numit *Nautilus*. Pentru a naviga pe *Internet*, aveți la dispoziție *browser*-ul *Mozilla* care a ajuns la versiunea 0.9.6.

Alături de acestea este prezent un set complet de aplicații de birou *Sun StarOffice* versiunea 5.2.

Sistemul de operare *RedHat Linux* 7.2 are la bază *kernel*-ul *Linux* 2.4.7. Acesta poate fi foarte ușor actualizat cu ultima versiune disponibilă a *kernel*-ului care era, la închiderea ediției, 2.4.16.

Recent, a început dezvoltarea următoarei generații a *kernel*-ului *Linux*, dar versiunea 2.6 a acestuia nu va apărea prea curând. Momentan este disponibilă versiunea 2.5 care nu este considerată stabilă.

Trebuie amintit faptul că pentru versiunile considerate stabile al doilea număr trebuie să fie par. De exemplu, versiunile 2.2.x sau 2.4.x sunt stabile, în timp ce stabilitatea versiunilor 2.3.x sau 2.5.x nu este garantată.

Instalare

Pentru a putea instala sistemul de operare *RedHat Linux* 7.2 aveți la dispoziție cel puțin trei posibilități. Una dintre ele ar fi crearea unei dischete de *boot*, folosind utilitarul `rawrite` care vă ajută să creați pe dischetă imaginea corespunzătoarea modalității de instalare a sistemului (de pe discul hard, *CD-ROM*, rețea etc.). După crearea dischetei de *boot*, trebuie repornit sistemul (configurat pentru *boot*-are de pe dischetă) cu discheta prezentă în unitate.

În cazul în care dețineți *CD*-ul *boot*-abil cu programul de instalare și *BIOS*-ul sistemului vostru permite încărcarea sistemului de operare de pe un astfel de *CD*, crearea dischetei respective nu mai este necesară.

Ultima variantă vă oferă posibilitatea începerii procesului de instalare direct de pe *CD*, fără a mai reporni sistemul, dar numai dacă sistemul vostru de operare suportă platforma *Dos* în mod real (nu lansat de *Windows*), deoarece trebuie să executați fișierul de comenzi `autoboot.bat` prezent pe primul *CD* de instalare.

Programul de instalare are o interfață grafică, mult mai ușor de utilizat decât interfața text prezentă la primele versiuni ale acestui sistem de operare.

În timpul procesului de instalare a sistemului de operare trebuie ales discul și partiția pe care acest sistem va fi instalat.

În această fază aveți la dispoziție ultima versiune a utilitarului *Disk Druid*, care este folosit la partiționarea discului hard. Această versiune permite crearea de partiții de tip *ext3*.

Datorită îmbunătățirilor aduse, instalarea sistemului *RedHat* 7.2 nu va ridica multe probleme.

Configurare

În timpul procesului de instalare, va trebui să realizați diferite setări cum ar fi cele referitoare la rețea, lista utilizatorilor care vor folosi sistemul, tipul plăcii video și rezoluția utilizată, tipul monitorului, rata de *refresh* a acestuia etc.

FIFA 2002

Seria FIFA a celor de la EA Sports este de mult timp cunoscută pentru grafica excelentă, animațiile bine realizate și realismul jocului. Recent, a apărut ultimul joc din această familie care a adus o serie de facilități noi, multe dintre ele fiind sugerate de către fanii jocului. FIFA 2002 este cel mai bun simulator de fotbal creat vreodată, motiv pentru care a devenit jocul lunii decembrie.

Introducere

Ideea care stă la baza seriei *FIFA* este foarte simplă: simularea unui joc de fotbal. Primele versiuni au impresionat prin calitatea simulării, realismul jocului fiind destul de ridicat.

Pe parcursul anilor, jocul a fost îmbunătățit, fiecare nouă versiune aducând o serie de noutăți care măreau atractivitatea jocului.

Noutăți

Cea mai importantă noutate introdusă de *FIFA 2002* este controlul configurabil. Până la versiunea anterioară, jucătorii erau obligați să folosească tastele predefinite. Producătorii nu au permis modificarea tastelor prin intermediul cărora se efectuează diferite acțiuni. Acesta era unul dintre puținele motive pentru care era criticată această serie. *FIFA 2002* înlătură acest neajuns; acum jucătorii au libertatea să aleagă tastele corespunzătoare diferitelor acțiuni.

O altă noutate introdusă de ultima versiune este sistemul de pase. În *FIFA 2001* apăreau săgeți ale căror culori indicau șansa ca mingea să ajungă la un coechipier sau la un adversar. O săgeată verde indica faptul că este foarte probabil ca pasa să ajungă la destinație, în timp ce săgeata roșie semnala faptul că există foarte puține șanse ca mingea să nu ajungă la adversar. Culoarea galbenă reprezenta situația în care era posibil ca mingea să ajungă la coechipier, dar existau șanse ca ea să fie interceptată de un adversar.

În *FIFA 2002* aveți posibilitatea de a le cere coechipierilor să înainteze. Pentru aceasta va trebui să apăsați o singură tastă. Vor apărea linii punctate care vor indica traiectoriile acestora. Astfel, puteți să nu pasați direct coechipierului, ci să trimiteți mingea în fața acestuia, urmând ca el să o preia.

Forța cu care este trimisă mingea în momentul unei pase, unui lob sau al unui șut este determinată de durata în care mențineți apăsată tasta corespunzătoare. Apare și un indicator vizual care reprezintă forța cu care este trimisă mingea.

Echipe

EA Sports a obținut licențe pentru a putea include în joc 75 de echipe naționale și echipele din 16 ligi profesioniste cum ar fi *Premier League*, *Bundesliga* etc.

Jocul se axează pe calificarea la *Cupa Mondială* care se va desfășura în vara anului 2002 în *Japonia* și *Coreea de Sud*. Veți putea retrăi emoțiile meciurilor echipei *României* și, foarte probabil, veți rejuca barajul de calificare cu *Slovenia*. Din păcate, victoriile voastre vor rămâne virtuale...

Sistemul necesar

Configurația minimă indicată de *Electronic Arts* include un procesor *Intel Pentium MMX* cu frecvența de cel puțin 200 MHz, unitate CD cu viteza de cel puțin 4×, placă video cu accelerator 3D și cel puțin 4 MB de memorie și 80 MB spațiu liber disponibil pe disc.

FIFA 2002 este acum disponibil doar pentru platforma *Windows*. Sistemele de operare acceptate sunt *Windows '95*, *Windows '98*, *Windows Me*, *Windows 2000* și *Windows XP*. Pentru rularea jocului este necesar și pachetul *DirectX 7.0a*.

Dacă folosiți *Windows XP* veți avea nevoie de 128 MB de memorie. Pentru *Windows 2000* sunt necesari doar 64 MB, iar pentru celelalte sisteme de operare suportate sunt suficienți 32 MB.

Configurația recomandată este însă: procesor *Intel Pentium II*, unitate CD cu viteza de cel puțin 8×, placă video cu accelerator 3D și cel puțin 8 MB de memorie și 500 MB spațiu disponibil pe disc.

Pentru *Windows XP* și *Windows 2000* se recomandă 128 MB de memorie, iar pentru celelalte sisteme de operare recomandarea este de 64 MB.

MOȘ Crăciun

În apropierea sărbătorilor de iarnă, Internet-ul a fost invadat de site-uri dedicate acestei perioade a anului. Bineînțeles, au apărut și câteva site-uri românești de acest tip. În acest număr vă vom prezenta unul dintre acestea și anume: www.moscraciun.ro.

Introducere

Imediat după încărcarea *site*-ului veți observa ca pagina este, de fapt, doar o redirectare spre **www.sarbatori.acasa.ro**.

Totuși, *site*-ul pe care îl găsiți aici merită vizitat. Veți găsi o mulțime de informații referitoare la sărbătoarea Crăciunului, dar nu numai atât.

Secțiuni

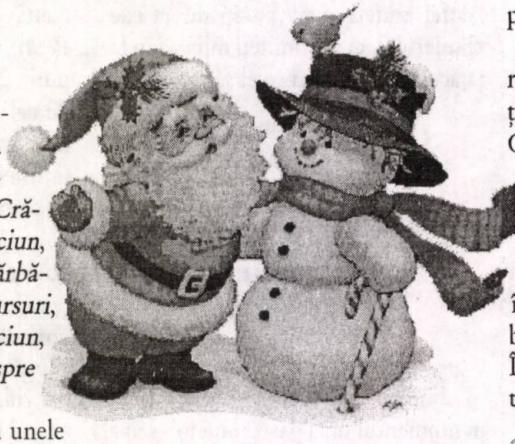
Meniul pe care îl aveți la dispoziție vă permite să alegeți una dintre secțiunile *site*-ului. Acestea sunt: *Totul despre Moș Crăciun, Scrisoare pentru Moș Crăciun, Pomul de Crăciun, Bucate de sărbători, Muzică, Cadouri, Concursuri, Chat cu Moșu', Jocuri de Crăciun, Trimite o felicitare* și *Păreri despre site*.

Din păcate, pentru a accesa unele dintre acestea, trebuie să fiți utilizatori înregistrați, deci trebuie să completați un formular.

În continuare vă vom prezenta, pe scurt, câteva dintre secțiunile disponibile.

Secțiunea *Totul despre Moș Crăciun* conține o mulțime de povestiri despre sărbătorile de iarnă sau *Moș Crăciun*. De asemenea, veți găsi și informații referitoare la modul în care este sărbătorit Crăciunul în alte țări ale lumi cum ar fi *Marea Britanie*, *Statele Unite ale Americii*, *Ungaria* etc.

În secțiunea dedicată pomului de Crăciun se află, pe lângă descrierea acestui obicei, o prezentare completă a simbolisticii bradului în tradiția românească.



Acesta apare în toate momentele semnificative ale vieții; veți găsi detalii despre bradul de naștere, bradul de nuntă sau cel funerar, dar nu numai. De asemenea, sunt prezentate diferite ornamente care nu pot lipsi de pe pomul de Crăciun.

Secțiunea *Muzică* vă oferă informații despre cântecele de Crăciun, despre colindele românești sau cele din alte țări.

Una dintre cele mai interesante secțiuni este cea dedicată cadourilor. Pe lângă obișnuitele sfaturi despre modul în care puteți alege cadourile, veți găsi aici și regulile de bază pentru cei care oferă un cadou, dar și regulile pe care trebuie să le respectați atunci când primiți un cadou.

Există și o secțiune dedicată dialogului cu *Moș Crăciun*. Aici veți putea să îi cereți cadourile pe care doriți să le primiți. (Nu înseamnă și că le veți primi, dar merită să încercați.)

Site-ul vă oferă posibilitatea să vă relaxați cu câteva jocuri ale căror acțiuni sunt inspirate de sărbătoarea Crăciunului. În unul dintre jocuri va trebui să încercați să prindeți cât mai multe cadouri, dar să evitați globurile. Într-un alt joc sunteți invadați de spiriduși care încearcă să vă fure cadourile care trebuie să ajungă la copii cu orice preț. În total, există șase jocuri care vă așteaptă.

Încheiere

Profităm de această ocazie pentru a ura *"Sărbători fericite!"* tuturor cititorilor revistei noastre.

PROBLEME propuse pentru REZOLVARE

Vă prezentăm enunțurile celor șase probleme propuse spre rezolvare concurenților de la Olimpiada Balcanică de Informatică, desfășurată în perioada 8-13 septembrie 2001 în stațiunea Durres din Albania.

P080101: Mozaic

Presupunem că avem un mozaic într-o zonă dreptunghiulară H formată din $M \times N$ celule dispuse în M linii și N coloane. În acesta este încorporat un model original. Acesta se află într-o altă zonă dreptunghiulară O, formată din $P \times Q$ celule dispuse în P linii și Q coloane. Toate celulele din cele două zone sunt colorate în alb sau negru ($1 < M, N, P, Q \leq 100$).

Toate celulele negre ale zonei dreptunghiulare O fac parte din mozaic. Toate celulele albe situate între două celule negre pe una din direcțiile orizontală, verticală sau diagonală fac parte din mozaic.

Dacă se dă mozaicul original în zona O, scrieți un program care determină dacă în zona dreptunghiulară H se regăsește o copie identică a acestuia (având aceeași orientare, fără a fi rotit). În caz afirmativ, veți scrie YES pe prima linie a fișierului de ieșire, iar în linia a doua două numere întregi care reprezintă coordonatele primei celule a mozaicului. Prima celulă a mozaicului este considerată a fi cea care are coordonata cea mai mică a liniei și, în cadrul liniei respective, coordonata cea mai mică a coloanei.

Date de intrare

Fișierul de intrare **MOSAIC.IN** conține pe prima linie două numere întregi P și Q, reprezentând dimensiunile zonei dreptunghiulare în care se află mozaicul original O.

Următoarele P linii conțin fiecare câte Q numere întregi, cu valori 0 sau 1. Valoarea 0 reprezintă o celulă de culoare albă, iar valoarea 1 reprezintă o celulă de culoare neagră. Numerele sunt separate prin câte un spațiu.

Următoarea linie conține două numere întregi M și N care reprezintă dimensiunile zonei dreptunghiulare H, în care se presupune că există o copie a mozaicului. Urmează apoi M linii, conținând fiecare câte N numere întregi din mulțimea {0, 1}.

Date de ieșire

Fișierul de ieșire **MOSAIC.OUT** va avea una sau două linii. În prima linie a fișierului se va scrie YES sau NO, în funcție de existența unei copii a mozaicului din zona O în zona dreptunghiulară H.

Dacă prima linie conține YES, în linia a doua vor fi scrise două numere întregi, separate printr-un spațiu, reprezentând coordonatele primei celule negre a copiei mozaicului.

Exemple

MOSAIC.IN

```
7 8
0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0
0 0 1 1 1 1 0 1
0 1 1 0 0 1 1 0
0 0 1 1 1 1 0 1
0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0
8 9
1 1 0 0 1 1 1 1 0
0 0 0 0 0 0 1 1 1
0 0 0 1 1 0 0 1 0
1 1 1 1 1 1 0 1 0
1 1 1 0 0 1 1 0 1
1 0 1 1 1 1 0 1 0
1 1 0 1 1 0 0 0 1
1 1 0 0 0 0 1 1 0
```

MOSAIC.OUT

```
YES
3 4
```

MOSAIC.IN

```
3 3
0 1 0
1 0 1
0 1 0
3 3
0 1 0
1 1 1
0 1 0
```

MOSAIC.OUT

```
NO
```

Timp maxim de execuție/test: 1 secundă

P080102: Șirul secret

Trebuie să descoperiți un șir secret format din patru cifre din mulțimea {0, 1, ..., 9}. Pentru a descoperi acest șir, la început veți presupune că este un șir oarecare. Apoi veți apela rutina `ShowFit()`, care va compara acest șir cu cel secret și vă va arăta cum se *potrivesc* cele două șiruri. Rutina are doi parametri (`rp` și `wp`) transmiși prin referință; după executarea rutinei, ei vor avea valori cuprinse între `0` și `4`. Parametrul `wp` va indica numărul de cifre comune celor două șiruri, dar care nu ocupă aceleași poziții în aceste șiruri. Parametrul `rp` va indica numărul de cifre comune celor două șiruri, care ocupă aceleași poziții.

De exemplu, dacă șirul secret este `1733`, atunci pentru șirul presupus `2373` rutina `ShowFit()` va furniza valorile `wp=2` (deoarece în ambele șiruri există un `7` și un `3`, dar nu pe aceleași poziții), și `rp=1` (deoarece există un `3` care ocupă în ambele șiruri aceeași poziție).

Scrieți un program care, utilizând rutina `ShowFit()`, descoperă șirul secret. Procedura nu poate fi apelată decât de cel mult 10 ori.

Date de ieșire

Fișierul de ieșire **`SECRET.OUT`** va avea o singură linie care va conține șirul secret.

Exemplu

Șiruri presupuse	Valorile `wp` și `rp`
1234	1 1
2134	0 2
1134	1 2
1131	0 3
2131	0 3
3131	0 4

deci `3131` este șirul secret.

`SECRET.OUT`

```
3131
```

Instrucțiuni pentru programatorii în Pascal

Programul vostru trebuie să conțină declarația **`uses`** `Show;`

Apelul procedurii `ShowFit()` trebuie să fie de forma:

```
ShowFit(guess,wp,rp);
```

unde variabila `guess` are tipul **`string`**`[4]`, iar variabilele `wp`, `rp` au tipul `Integer`.

Instrucțiuni pentru programatorii în C/C++

Va trebui să creați un proiect în care veți include fișierul sursă și fișierul **`SHOWFIT.OBJ`** care vă este pus la dispoziție.

În programul sursă trebuie să includeți fișierul header **`SHOWFIT.H`**

Apelul funcției `ShowFit()` trebuie să fie de forma:

```
showfit(guess,&wp,&rp);
```

unde parametrul `guess` are tipul **`char`**`[4]`, iar parametrii `wp` și `rp` au tipul **`int`** și sunt transmiși prin referință.

Timp maxim de execuție/test: 1 secundă

P080103: Vizită

Un birou de consultanță este organizat astfel: toți clienții stau la o coadă de așteptare, iar consultarea se realizează conform principiului *primul sosit primul servit*. Consultantul unui client este determinat de sexul clientului anterior (femeie - `F` sau bărbat - `M`). Astfel, consultantul C_i, după ce primește un client, în funcție de sexul acestuia (`F`, `M`) direcționează următorul client la consultantul C_j, dacă clientul curent este femeie (`F`) sau la consultantul C_k, dacă clientul curent este bărbat (`M`). După aceea, consultantul ales va servi următorul client.

Un client isteț a descoperit că există cozi în care este posibil ca indiferent de consultantul cu care începe consultarea, parcurgând toți clienții dinaintea sa, el să fie consultat de același consultant, C_m.

Scrieți un program care determină dacă acest lucru este posibil și, în caz afirmativ, determină o coadă de lungime minimă care reprezintă clienții aflați în fața clientului isteț și consultantul care îl va trata.

Dacă o astfel de coadă nu există, programul vostru va furniza mesajul: `'Impossible'`.

Consultanții sunt reprezentați folosind numere întregi cuprinse între 1 și N.

Date de intrare

Fișierul de intrare **`VISIT.IN`** conține pe prima linie un număr întreg N ($2 < N \leq 40$), care reprezintă numărul de consultanți din birou.

Fiecare din următoarele N linii conțin câte trei numere întregi. Primul număr întreg reprezintă consultantul curent, al doilea întreg reprezintă consultantul care va continua consultarea în cazul în care clientul care a fost consultat a fost femeie, iar al treilea întreg reprezintă consultantul care va continua consultarea în cazul în care clientul consultat a fost bărbat.

Date de ieșire

Fișierul de ieșire **`VISIT.OUT`** va conține trei linii, dacă o astfel de coadă există, și o singură linie în caz contrar.

În cazul în care coada există, prima linie a fișierului de ieșire va conține lungimea cozii, pe linia a doua se va afla un șir de caractere `M` și `F` reprezentând coada, iar linia a treia va conține consultantul care va consulta clientul isteț.

În cazul în care coada nu există, fișierul va conține o singură linie care va conține mesajul `'Impossible'`.

Exemple

`VISIT.IN`

```
5
1 2 4
2 4 1
3 2 3
4 1 4
5 3 5
```

`VISIT.OUT`

```
4
FFMM
4
```

Timp maxim de execuție/test: 5 secunde

P030104: Joc

Doi jucători joacă următorul joc: se dă o grămadă care conține N bețe ($4 \le N \le 2000$); jucătorii pot lua din grămadă, pe rând, un număr de bețe; jucătorul care ia ultimul băț câștigă jocul.

Jocul se desfășoară respectându-se următoarele reguli:

- Jucătorul care începe jocul poate lua orice număr L_1 de bețe, dar nu întrega grămadă ($1 \le L_1 \le N - 1$).
- Apoi, cei doi jucători iau pe rând unul sau mai multe bețe, dar nu mai mult decât dublul numărului de bețe luate de celălalt jucător la mutarea precedentă. Astfel, jucătorul care este la mutare poate lua din grămadă L_2 bețe, unde $1 \le L_2 \le 2 \cdot L_1$ (L_1 este numărul de bețe luat de celălalt jucător la mutarea precedentă).

Scrieți un program care joacă acest joc împotriva calculatorului și câștigă. În cazul în care strategia voastră este corectă și sunteți jucătorul care începe jocul, veți câștiga întotdeauna.

La început se va citi din fișierul **GAME.IN** numărul N al bețelor din grămadă, apoi trebuie să se determine numărul L_1 al bețelor pe care le extrage din grămadă primul jucător (programul vostru).

Folosind aceste valori, se va apela rutina `move(N1,L1,N2,L2);`. Variabila `N1` reprezintă numărul de bețe care se află în grămadă înaintea efectuării mutării, iar variabila `L1` reprezintă numărul de bețe pe care primul jucător le extrage din grămadă. După executarea rutinei, variabila `N2` va indica numărul de bețe care rămân în grămadă după mutarea primului jucător (programul vostru), iar variabila `L2` va indica numărul de bețe pe care le extrage al doilea jucător (calculatorul) din grămadă. Se observă că avem întotdeauna `N2=N1-L1` și că la primul apel `N1=N`.

După apelarea rutinei, va trebui să determinați următorul număr al bețelor care vor fi extrase, precum și numărul bețelor care au rămas după mutarea calculatorului. Evident, acest ultim număr va fi întotdeauna `N1=N2-L2`. Veți apela din nou rutina `move()`, și acești pași se vor repeta până când nu mai există nici un băț în grămadă. Jucătorul care extrage ultimul băț câștigă jocul.

Exemplu

GAME.IN

```
10
```

Mutarea programului		*Mutarea calculatorului*	
N1	L1	N2	L2
10	2	8	1
7	2	5	1
4	1	3	1
2	2	0	

Notă

Programul vostru trebuie să respecte restricțiile jocului. În cazul în care o mutare nu este corectă, programul se va încheia automat. Rutina `move()` verifică fiecare mutare. În cazul în care strategia folosită nu este optimă, calculatorul va câștiga jocul.

Instrucțiuni pentru programatorii în Pascal

Programul vostru trebuie să conțină declarația **uses** `moveun;`

Apelul procedurii `move()` trebuie să fie de forma:

```
move(N1,L1,N2,L2);
```

unde variabilele `N1`, `L1`, `N2` și `L2` au tipul `Integer`.

La început se va apela, o singură dată, procedura `newgame`. Aceasta nu are nici un parametru.

Instrucțiuni pentru programatorii C/C++

Va trebui să creați un proiect în care veți include fișierul sursă și fișierul **MOVEUN.OBJ** care vă este pus la dispoziție.

În programul sursă trebuie să includeți fișierul header **MOVEUN.H** precum și declarațiile:

```
void newgame(void);
void move(int,int,int&,int&);
```

Apelul funcției `move()` trebuie să fie de forma:

```
move(N1,L1,&N2,&L2);
```

unde variabilele `N1`, `L1`, `N2` și `L2` au tipul **int**.

La început se va apela, o singură dată, funcția `newgame()`.

Timp maxim de execuție/test: 1 secundă

P030105: Orar

Un laborator acceptă vizitatori de la firmele interesate de utilizarea aparaturii proprii, doar două ore pe zi, într-un interval fix de zile.

Fiecare firmă trebuie să-și determine propriul orar pentru utilizarea laboratorului exact două ore pe zi. Aceste două ore pot fi în prima perioadă a zilei, în a doua perioadă sau în a treia perioadă.

Orarul este fixat, de la început, pentru toată perioada celor N zile și trebuie să respecte următoarele condiții:

- dacă o firmă selectează într-o zi a treia perioadă, atunci este obligată ca în următoarea zi să selecteze prima perioadă;
- prima perioadă poate fi selectată numai dacă în ziua precedentă a fost selectată perioada a treia.

Pentru un număr de zile N dat ($1 \le N \le 44$), scrieți un program care calculează numărul M_N al posibilităților de întocmire a orarului.

Exemplu

Pentru $N = 4$, cele $M_N = 5$ posibilități de întocmire a orarului sunt:

Ziua	**1**	**2**	**3**	**4**
Orar 1	2	2	2	2
Orar 2	3	1	3	1
Orar 3	2	3	1	2
Orar 5	2	2	3	1
Orar 5	3	1	2	2

Se observă că, respectând condițiile, nu se poate programa prima perioadă în prima zi și a treia perioadă în ultima zi.

Date de intrare
Fișierul de intrare **SCHEDULE.IN** conține o singură linie pe care se află numărul întreg N, care reprezintă numărul de zile.

Date de ieșire
Fișierul de ieșire **SCHEDULE.OUT** va conține o singură linie pe care se va afla numărul întreg M_N, care reprezintă numărul de posibilități de întocmire a orarului.

Timp maxim de execuție/test: 1 secundă

P080106: Votare

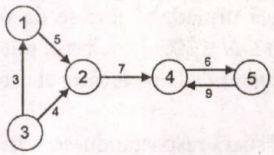
Într-o zonă care are N ($3 \le N \le 50$) localități, trebuie să fie construite centre de votare. Într-un centru de votare pot vota doar locuitorii din localitățile care sunt legate prin drumuri cu toate celelate localități ale căror locuitori votează în același centru.

Se cunosc localitățile și drumurile care le leagă. Drumurile sunt date într-o matrice D având elemente întregi, unde d_{ij} reprezintă lungimea drumului care leagă localitatea i de localitatea j, ($d_{ij} = 0$ are semnificația: nu există drum între localitatea i și localitatea j). Toate drumurile sunt cu sens unic.

Programul vostru trebuie să determine numărul minim K al zonelor de votare care pot fi formate.

După aceea, pentru fiecare zonă, trebuie determinate localitățile P_s ($s = 1, 2, ..., K$), în care trebuie construite centrele de votare, astfel încât suma lungimilor tuturor drumurilor de la fiecare localitate la acest centru să fie minimă. În cazul în care, pentru o zonă de votare există mai multe localități în care se poate construi centrul de votare care satisface condițiile de minim, se va determina una din ele.

Să considerăm zona reprezentată în figura următoare:



În acest caz există două zone de votare: (1, 2, 3) și (4, 5). Centrele de votare sunt în localitățile 1 și 5.

Date de intrare
Fișierul de intrare **VOTING.IN** conține pe prima linie numărul întreg N, care reprezintă numărul de localități. Următoarele N linii conțin fiecare câte N numere întregi, separate prin spații, care reprezintă lungimile drumurilor.

Date de ieșire
Fișierul de ieșire **VOTING.OUT** trebuie să conțină K linii; pe fiecare dintre ele se vor afla două numere întregi P și L, reprezentând localitatea în care se construiește centrul de votare și distanța minimă. Liniile vor fi ordonate crescător în funcție de numărul de ordine al localității.

Exemplu

VOTING.IN
```
5
0 5 0 0 0
5 6
0 0 4 7 0
3 0 0 0 0
0 0 0 0 6
0 0 0 9 0
```

VOTING.OUT
```
1 10
5 6
```

Timp maxim de execuție/test: 2 secunde

România la BOI 2001

La BOI 2001, România a obținut, pentru prima dată la un concurs internațional de programare, patru medalii de aur.

Elevii români care au realizat această performanță deosebită sunt:

Daniel Dumitran (București)
Bors Vencel (Oradea)
Victor Costan (București)
Vlad Dascălu (Bacău)

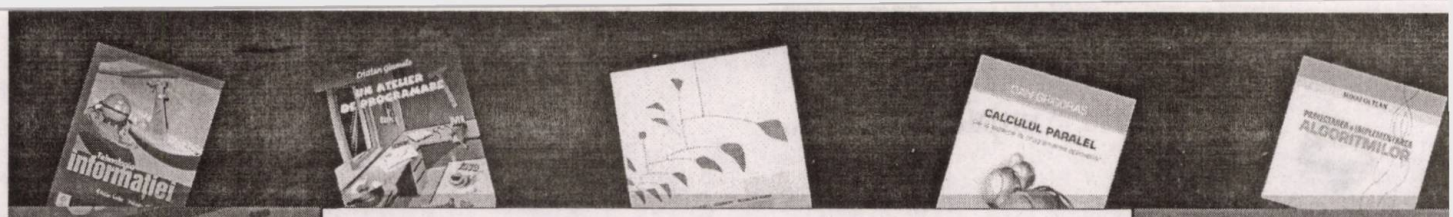
Delegația României l-a avut ca team leader pe dl. lect. univ. Ovidiu Domșa de la Universitatea "1 Decembrie 1918" din Alba Iulia și ca deputy leader pe dl. prof. Marinel Șerban de la Liceul "Grigore Moisil" din Iași.

Trebuie remarcat faptul că nici un alt concurent nu a mai obținut medalie de aur, elevii români clasându-se pe primele patru locuri. Evident, în clasamentul pe națiuni, România a ocupat primul loc, la mare distanță de Bulgaria, cea de-a doua clasată.

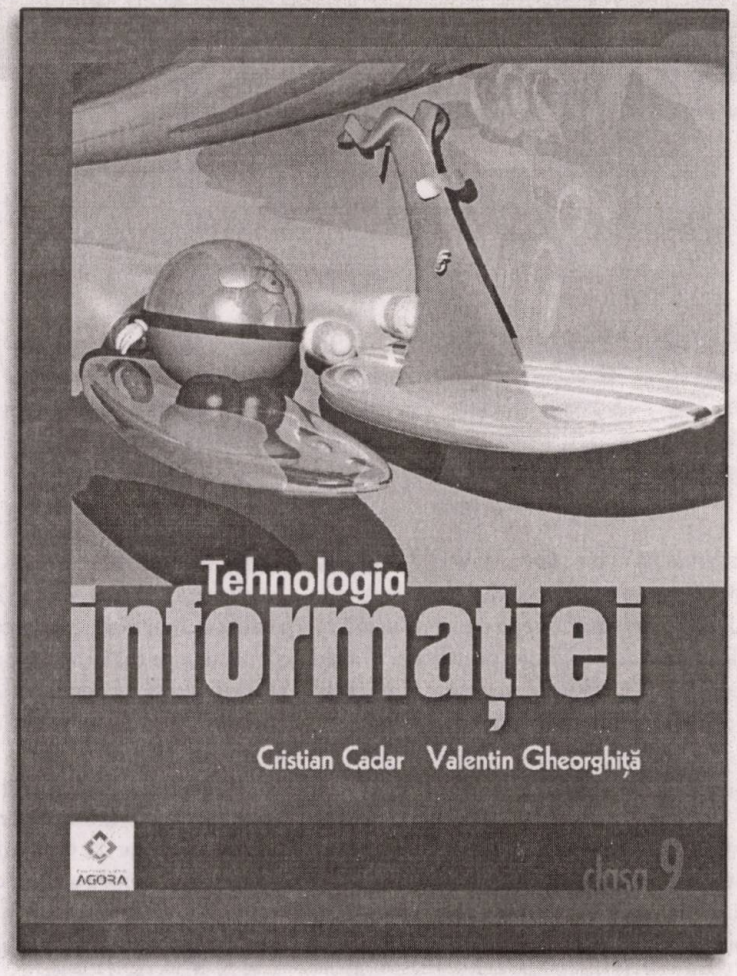
În cea de-a doua zi de concurs, trei dintre reprezentanții României au obținut punctajul maxim (300 de puncte), iar al patrulea a obținut "doar" 280 de puncte.

În final, punctajele concurenților români au fost de 580 (Daniel Dumitran), 560 (Victor Costan și Vencel Bors) și 520 (Vlad Dascălu) de puncte.

Concurentul clasat pe a cincia poziție a obținut 400 de puncte.



COMPUTER LIBRIS AGORA



Lucrarea de față își propune să pregătească cititorul în scopul utilizării unui calculator personal. Ea urmărește programa școlară, aprobată de Ministerul Educației Naționale, pentru disciplina "Tehnologia informației", disciplină în planul de învățământ a claselor a IX-a la profilul teoretic și pedagogic. Manualul corespunde și programei de la disciplina "Informatică - Tehnologii asistate pe calculator", profilul tehnologic.

Pentru parcurgerea acestei cărți nu sunt necesare cunoștințe prealabile în domeniu, termenii și noțiunile prezentate sunt explicate detaliat în carte. Exemplele sunt sugestive, cartea nu este încărcată inutil cu informații care depășesc programa amintită.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Un PARSER de expresii REGULATE

Alexandru Şuşu

Programul GRAPE, despre care vom vorbi în următoarele pagini, este un parser de expresii regulate şi a fost implementat de către autor împreună cu un colaborator (Daniel Cuclea, student la Universitatea Politehnica din Bucureşti), folosind limbajul C++. Sursele acestui program pot fi descărcate de la adresa www.ginfo.ro/revista/11_8/.

Pentru început, trebuie să avertizăm cititorii că articolul conţine o mulţime de cuvinte preluate din limba engleză, de care am considerat că este mai bine să nu ne dispensăm, pentru a vă familiariza cu termenii de specialitate. Un alt motiv la fel de important îl constituie inexistenţa unor traduceri acceptabile în limba română pentru unii termeni.

De ce expresii regulate?

În jurul anilor 1960, perioadă în care se puneau bazele primelor compilatoare, cercetătorii au observat că este mai simplu de realizat şi de portat un compilator dacă se proiectează în mai multe blocuri, fiecare bloc fiind specializat pe un anumit proces. Primul bloc al unui compilator poartă numele de ***analizor lexical*** şi are următoarele atribuţii:

- recunoaşte şi verifică, după o serie de criterii, validitatea "cuvintelor" (entităţi care se numesc *token*-uri) din codul sursă al programului scris în respectivul limbaj de programare (cuvintele cheie, identificatorii şi constantele numerice etc.) şi le trimite către următorul bloc din compilator;
- dacă analizorul lexical întâlneşte un *token* invalid (de exemplu, în cazul limbajelor *Pascal* sau *C/C++*, un identificator de funcţie care începe cu o cifră), atunci apelează o rutină de tratare a erorii. Trebuie remarcat faptul că, dacă blocul de analiză lexicală nu găseşte nici o eroare, aceasta nu înseamnă că programul este corect. În această fază se aplică doar o parte a criteriilor de validitate pe care un program trebuie să le respecte.

S-a dorit să se proiecteze un analizor lexical într-un mod cât mai elegant. Pentru aceasta s-au pus bazele unui aparat teoretic care să ajute la recunoaşterea "cuvintelor" programelor. Poate vă veţi întreba care este motivul unor astfel de complicaţii, deoarece s-ar putea folosi bibliotecile şi funcţiile pe care le avem deja la dispoziţie. În cazul în care ar trebui să recunoaştem cuvinte ca "for", "while", este adevărat că ar fi suficientă apelarea unei funcţii de tipul `strcmp()`.

Dificultatea apare în situaţia în care trebuie să recunoaştem un identificator (de exemplu, o funcţie definită de utilizator), deoarece numărul acestor identificatori poate fi foarte mare. Pentru a accepta identificatorul ca fiind valid trebuie să verificăm dacă primul caracter este literă (sau caracterul `'_'`) şi dacă restul caracterelor sunt alfanumerice (sau alte caractere speciale admise, cum ar fi `'_'`, `'$'`). Totuşi, putem implementa rutine destul de simple care recunosc şirurile de caractere care sunt identificatori de funcţii.

Lucrurile încep să se complice mai mult în cazul constantelor numerice: constantele reprezentate în virgulă mobilă pot fi scrise sub formă ştiinţifică cu exponent (de exemplu, `1.894E-3`). Recunoaşterea constantelor numerice presupune o rutină în care se verifică destul de multe condiţii: dacă există caracterul `'.'`, atunci el trebuie să fie urmat de cel puţin o cifră; dacă există caracterul `'E'` (sau `'e'`) acesta trebuie să fie urmat de un număr întreg etc.

Fiecare dintre cazurile de mai sus au câteva trăsături comune. Astfel, în loc să se trateze separat fiecare caz, se poate concepe o bibliotecă de funcţii de recunoaştere mai puternică, dar şi mai complexă, care să permită tratarea fiecărui caz prin particularizare. Se poate aduce următorul

contraargument acestei soluții: *o implementare mai generală este cu siguranță mai puțin eficientă decât una care se folosește de toate particularitățile problemei.* Fără îndoială, afirmația este adevărată dar, în cazul programelor de o complexitate ridicată, se recomandă folosirea unei soluții generale (care promite un ordin de complexitate *similar-polinomial* cu alte soluții particularizate), având în vedere riscul mai mic de apariție a unor erori de programare.

Pentru tratarea unificată a analizei lexicale s-a dezvoltat ***teoria expresiilor regulate***, la ora actuală existând în practică o suită impresionantă de aplicații care folosesc din plin acest tip de expresii.

Ce este o expresie regulată?

Expresiile regulate se folosesc pentru a căuta șiruri de caractere specificate prin informații incomplete privind modelul care trebuie găsit. Ele descriu parțial șirurile de caractere pe care dorim să le căutăm. În cazul în care un șir corespunde descrierii date de expresia regulată spunem că s-a găsit o ***potrivire a expresiei regulate peste șirul respectiv.*** Cel mai simplu exemplu este cel al folosirii comenzilor *DOS*, de tipul `"dir *.txt"` sau `"dir a?b.txt"`. *Wildcard*-urile folosite în descrierea numelor fișierelor sunt un exemplu nestandard de expresii regulate.

Expresia `"*.txt"` descrie incomplet o clasă largă de șiruri de caractere, anume cele care au ca sufix șirul `".txt"`, adică cele care au extensia `"txt"`. Un nume de fișier care corespunde acestei expresii este `"setup.txt"`. Expresia `"*.txt"` nu se potrivește peste numele de fișier `"io.sys"` (pentru că extensia este `"sys"`).

Expresia `"a?c.txt"` descrie incomplet o altă clasă de șiruri de caractere, și anume cele care au primul caracter `'a'`, urmat de orice caracter, urmat de șirul `"b.txt"`. Așadar, `"abc.txt"` și `"acc.txt"` corespund expresiei, dar șirul `"acb.txt"` nu.

În concluzie, o expresie regulată poate descrie o mulțime foarte mare de șiruri de caractere. Această mulțime se numește ***limbajul*** respectivei expresii. Ne punem acum problema cum se scriu corect aceste expresii regulate. Trebuie avut în vedere următorul aspect: mulțimea tuturor expresiilor regulate (valide, bineînțeles) alcătuiesc limbajul expresiilor regulate; însă la rândul lor, fiecare expresie descrie un limbaj.

Am dori să vă prezentăm fundamentul mecanismului de rulare al programului `GRAPE`: fie o expresie regulată și un text; `GRAPE` va verifica validitatea expresiei (dacă nu este validă va returna un mesaj de eroare) iar, în cazul în care expresia este validă, va returna cel mai lung subșir de caractere din text care se potrivește peste expresia respectivă.

Există mai multe variante de limbaje de expresii regulate. Probabil cel mai simplu dintre acestea este sistemul de specificare a fișierelor cu *wildcard*-uri din *DOS* (sau din *shell*-ul *Unix*). Un limbaj mult mai complex, cu o putere mai mare de exprimare este cel descris în standardul `POSIX 1003.2` (*POSIX - Portable Operating System unIX* - este o specificație dezvoltată pentru standardizarea interfețelor sistemelor de operare de tip *UNIX*), care descrie două limbaje de expresii regulate: unul *"basic"* (care este vechi, dar este păstrat din motive de compatibilitate cu aplicațiile mai vechi) și unul extins.

tabelul 1

Operator	Tip	Simbol	Descriere
concatenare	unar	*juxtapunere*	dacă doi operanzi sunt adiacenți în expresie, atunci avem o potrivire dacă și numai dacă două subșiruri de caractere corespunzătoare celor doi operanzi sunt adiacente în text, în ordinea corespunzătoare; de exemplu, expresia `"AB"` se potrivește peste șirul de caractere `"AB"` și nu se potrivește peste nici un alt șir de caractere
sau	binar	`'|'`	ne permite să specificăm mai multe alternative care pot fi luate în considerare în cazul căutării unei potriviri; expresia `"A|B"` se potrivește doar peste două șiruri de caractere: `"A"` și `"B"`; expresia `"C(A|U)(I|L)"` se potrivește doar peste următoarele patru șiruri de caractere: `"CAI"`, `"CAL"`, `"CUI"` și `"CUL"`
Kleen-closure	unar	`'*'`	aplicat unui operand, indică faptul că respectivul operand poate apărea concatenat cu el însuși de oricâte ori (numărul aparițiilor poate fi și zero); expresia `"A*"` se potrivește peste orice șir de caractere care conține doar caractere `'A'` (inclusiv șirul vid); expresia `"(A|B)*"` se potrivește cu orice șir care conține doar caractere `'A'` și `'B'` (inclusiv șirul vid), de exemplu `"A"`, `"B"`, `"ABBA"`, `"BABA"` etc
paranteze	unar	`'('`, `')'`	prin gruparea simbolurilor este posibilă schimbarea priorității operatorilor
orice caracter	unar	`'.'`	expresia `"A."` se potrivește peste `"AB"`, `"AC"` etc; se observă că acest operand este echivalent cu expresia formată din disjuncția tuturor celorlalți operanzi prezentați, deci acesta nu este un operand elementar

Operanzii sublimbajului expresiilor regulate care este tratat în GRAPE

O expresie regulată este foarte asemănătoare din punct de vedere constitutiv cu o expresie aritmetică, ambele fiind compuse din operatori și operanzi.

În programul GRAPE am tratat un *sublimbaj* al limbajului expresiilor regulate extinse. Acest sublimbaj conține cei mai importanți operatori și operanzi. Operatorii sunt prezentați în tabelul 1.

Am putea să încheiem aici descrierea limbajului, dar îl vom îmbogați cu încă un element util. Să presupunem că dorim să căutăm într-un text o linie care conține caractere `'*'`. Trebuie să găsim o metodă pentru a face distincție între operandul (caracterul) `'*'` și operatorul *Kleene-closure* `'*'`. O soluție elegantă o reprezintă folosirea unui "***escape char***" - `'\'` care anulează semnificația specială a caracterului care urmează. Introducem așadar în limbaj mai mulți operanzi formați din perechea de caractere `'\'` și unul din caracterele speciale (operatorii) prezentați anterior, care transformă caracterul special în caracter obișnuit care va trebuie potrivit. Apare întrebarea: *cum se poate reprezenta operandul care se potrivește cu caracterul* `'|'`? Răspunsul este simplu: *îl vom prefixa cu un* ***escape char***. Reprezentarea acestui operand va fi `"\\"`.

Așadar, operanzii și operatorii din care sunt formate expresiile regulate sunt:

Simbol	Semnificație
`'A'-'Z','a'-'z'`	*litere*
`'0'-'9'`	*cifre*
`'.'`	*orice caracter*
`'(',')'`	*paranteze*
`'|'`	*disjuncție*
`'*'`	*închidere Kleene*
`"\(","\)","\*","\.","\\"`	*modificator*

Atribuirea acestei semnificații caracterului `'\'` permite reprezentarea caracterelor `'('`, `')'`, `'*'` și `'.'` fără semnificația lor specială, astfel încât să poată fi folosiți ca simpli operanzi (caractere care trebuie potrivite peste text). Pentru a putea folosi și caracterul `'\'` pe post de operand a fost introdus și operatorul `"\\"`.

În standardul POSIX 1003.2 mai există o serie de operanzi și operatori utili, care nu sunt implementați în programul GRAPE. Câțiva dintre aceștia sunt prezentați în tabelul 2.

Simplul fapt că știm care sunt caracterele care pot compune o expresie regulată nu înseamnă că am descris complet limbajul expresiilor regulate. Trebuie să stabilim niște reguli de interacțiune a acestora.

Pentru specificarea exactă a sintaxei limbajului expresiilor regulate se folosește un obiect matematic denumit ***gramatică independentă de context.***

În [5] se specifică faptul că o gramatică este un set de reguli privitoare, pe de o parte, la forma și modificarea cuvintelor (***morfologia***), iar, pe de altă parte, la îmbinarea cuvintelor în procesul comunicării (***sintaxa***). Tot în [5] se precizează că gramaticile se pot împărți în gramatici destinate marelui public și în gramatici adresate specialiștilor, acestea din urmă putând fi tradiționale sau moderne. Cele moderne pot fi de mai multe tipuri, printre care există și categoria gramaticilor matematice care ne interesează.

Gramaticile independente de context au fost introduse în 1956 de către ***Noam Chomsky***, în timp ce studia limbajele naturale. Adoptarea gramaticilor independente de context în cadrul științei calculatoarelor a fost realizată de către ***John Backus*** în timp ce lucra la o schiță a raportului limbajului de programare *Algol 60* (un "bunic" al limbajului *C*), pentru a descrie sintaxa respectivului limbaj.

Pentru aceasta s-a folosit o notație denumită la început *Backus-Normal Form* (*BNF*) (în 1964 ***Donald Knuth*** a propus într-o scrisoare ca notația să se numească ***Backus-Naur Form***, în onoarea lui ***Naur***, alt cercetător care a participat la redactarea raportului *Algol 60*). Notația *BNF* a gramaticii limbajului expresiilor regulate este:

```
<expression> ::= <term> |
                 <term>OR_OPERATOR<expression>
<term> ::= <factor> | <factor><term>
<factor>::=(<expression>) | ch |
           <factor>KLEENE_CLOSURE_OPERATOR,
```

unde `ch` reprezintă oricare dintre operanzi.

tabelul 2

Operator	Tip	Simbol	Descriere
listă de caractere	operand	`[N-M]`	permite o scriere mai compactă a expresiilor; de exemplu, expresia `"[0-9]"` este echivalentă cu expresia `"(0|1|2|3|4|5| 6|7|8|9)"`
complementare	operator	`'^'`	un exemplu în care se folosește acest operator este expresia `"[^0-9]"` care se potrivește peste toate șirurile de un caracter cu excepția celor care conțin o cifră
expresie interval	operator	`{N,M}`	expresia asupra căreia se aplică se poate repeta de cel puțin *N* ori și de cel mult *M* ori; de exemplu, expresia `"A{1,3}"` este echivalentă cu `"A|(AA)|AAA"`
semn de întrebare	operator	`'?'`	indică cel mult o apariție a operandului căruia i se aplică
back-referencing	operator	`'\n'`	potrivește peste subșirul potrivit anterior de a `n`-a subexpresie parantezată din expresia regulată. De exemplu, expresia `"(A*)B\1"` se potrivește peste șirul `"AAABAAA"`, dar nu se potrivește peste șirul `"AAABAAB"`

Operatori și operanzi specificați de standardul POSIX 1003.2

Am folosit notațiile KLEENE_CLOSURE_OPERATOR și OR_OPERATOR pentru operatorii ***Kleene-Closure*** și ***sau***, din dorința de a elimina ambiguitățile cu privire la semnificația caracterului '|'.

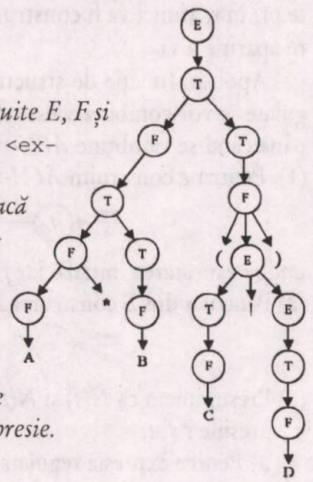
Această notație conține următoarele elemente:

- ***producții***, care sunt compuse din ***simboluri terminale*** (simboluri care aparțin limbajului: ch, OR_OPERATOR, KLEENE_CLOSURE_OPERATOR și parantezele rotunde), ***simboluri neterminale*** (simboluri interne gramaticii: <expression>, <term>, <factor>) și ***metasimboluri*** ("::=" și "|"). Metasimbolul "::=" are semnificația ***"este"***; el indică faptul că membrul stâng are, prin definiție, aceeași semnificație ca și membrul drept. Metasimbolul "|" reprezintă disjuncția celor doi termeni din stânga, respectiv din dreapta sa.
- neterminalul din membrul stâng din prima producție (<expression>) se numește ***simbol de start*** al gramaticii;

Cea de-a doua producție indică faptul că <term> reprezintă un <factor> sau un <factor> concatenat cu <term>. Observăm ca avem o definiție recursivă pe care, prin raționament inductiv, o putem extinde la următoarea definiție: <term> este o succesiune nevidă (cel puțin un element) de <factor>-i.

O metodă naturală de a descrie un proces format din mai multe derivări este ***arborele de derivare***. Arborii de derivare sunt foarte intuitivi, motiv pentru care vom da direct un exemplu de arbore.

Considerăm expresia A*B(C|D), al cărei arbore de derivare (***parse tree***) este prezentat în figură.



Nodurile interne încercuite E, F și T reprezintă neterminalii <expression>, <factor> *și* <term>. *Săgețile care pleacă de la un neterminal se îndreaptă spre simbolurile în care derivează respectivul neterminal.*

Arborele descrie toate etapele analizei sintactice având data respectiva expresie.

Din notația *BNF* a gramaticii se pot deduce următoarele proprietăți ale operatorilor:

- OR_OPERATOR are prioritatea mai mică decât operatorul concatenare, iar operatorul concatenare are prioritatea mai mică decât KLEENE_CLOSURE_OPERATOR.
- OR_OPERATOR și operatorul ***concatenare*** sunt asociativi la dreapta.

Exemple

Vom prezenta acum două exemple concrete de folosire a expresiilor regulate:

- clasa șirurilor de caractere care descriu numere reale de tipul 01, 12, 12.3, 12.3E4 poate fi descrisă folosind următoarea expresie regulată:

```
"[0-9]+(.[0-9]+)?(E(+|-)?digit+)?"
```

- expresia regulată care ar descrie numele valide de funcții în *C* este:

```
"(_|$|[a-zA-Z]) (_|$|[a-zA-Z0-9])*".
```

Cum se potrivește un șir peste o expresie regulată?

Pentru a rezolva sarcina potrivirii unui șir de caractere cu o expresie regulată, trebuie ca aceasta să fie transformată într-o formă mai accesibilă calculatorului. Teoria ne spune că pentru orice expresie regulată există cel puțin un *Automat Finit Nedeterminist* (*AFN*) și cel puțin un *Automat Finit Determinist* (*AFD*) care sunt echivalente cu expresia [2].

Programul GRAPE realizează conversia unei expresii regulate într-un *AFN* pe care îl folosește pentru a accepta șirurile de caractere care se potrivesc peste respectiva expresie.

Ce este un AFN?

În primul rând, un automat finit este un obiect matematic care are o mulțime finită de stări, evoluând de la o stare la alta pe baza unor date (caractere) de intrare.

Un *AFN* (întâlnit în literatura de specialitate și sub numele de ***syntax charts*** sau ***railroad normal form***) este un graf orientat în care nodurile reprezintă stările pe care le poate avea automatul la momente de timp distincte.

AFN-ul are desemnate o stare inițială (de *start*) și mai multe stări finale (*acceptoare*). Muchiile dintre stări se numesc tranziții, fiecare tranziție având asociat un caracter care indică faptul că se poate ajunge din starea curentă într-o altă stare, pe tranziția respectivă, doar atunci când caracterul din textul de intrare coincide cu caracterul tranziției. Printre caracterele care se pot asocia unei tranziții se introduce și caracterul nul - ε, prezența sa indicând faptul că tranziția se poate executa indiferent care este caracterul de la intrare.

Caracterul nedeterminist al *AFN*-ului este ilustrat de posibilitatea ca dintr-o stare să poată exista mai multe stări la care să se poată ajunge pe baza unui același caracter. Exact aceasta este diferența dintre un *AFN* și un *AFD*: un *AFD* are cel mult o stare în care se poate ajunge din starea curentă pe baza caracterului de intrare, pe când un *AFN* poate avea o mulțime de astfel de stări.

AFN-ul implementat în GRAPE acceptă șiruri de caractere din mulțimea: {'a'-'z', 'A'-'Z', '0'-'9', '.', '*', '\', '|', '(', ')'}. Mulțimea de caractere care se pot asocia unei tranziții este cea descrisă anterior reunită cu {ε}. În cazul unui șir de caractere de intrare pentru care dorim să determinăm dacă este acceptat de *AFN*, va trebui să ***executăm*** *AFN*-ul; această operație se realizează astfel:

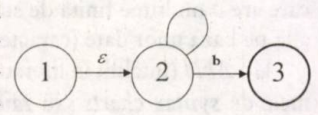
- inițial, *AFN*-ul se află în starea de start;
- când *AFN*-ul primește un caracter din textul de intrare, atunci el evoluează în funcție de caracterul primit astfel:

- dacă nu există tranziție care să iasă din starea curentă și care să conțină caracterul de intrare curent, atunci *AFN*-ul anunță exteriorul că nu s-a găsit potrivire între șirul de caractere de intrare și expresia regulată, și revine în starea inițială, așteptând alte secvențe de caractere;
- dacă există o tranziție din starea curentă pe baza caracterului de intrare curent, atunci *AFN*-ul execută tranziția respectivă; dacă există mai multe tranziții din starea curentă pe baza aceluiași caracter de intrare, atunci *AFN*-ul are puterea teoretică de a ghici tranziția care duce spre starea finală (spunem că *AFN*-ul este înzestrat cu puterea nedeterminismului, acest lucru înseamnă că reușește, având de ales dintre mai multe decizii, să o aleagă pe cea corectă); în continuare vom vedea cum se poate implementa o simulare de execuție a *AFN*-ului pe un calculator numeric, cu toate că pare paradoxal să se poată simula un automat nedeterminist, pe o mașină deterministă.

- când se ajunge într-una din stările finale, *AFN*-ul se resetează, anunță exteriorul că s-a găsit o potrivire și revine în starea inițială, așteptând alte secvențe de caractere.

Exemplu

Vom considera automatul finit nedeterminist din figura alăturată. Starea inițială a *AFN*-ului este 1. Cercul îngroșat care simbolizează starea 3 indică faptul că aceasta este o ***stare acceptoare***.



Presupunem că avem șirul de intrare `"ab"`. Inițial *AFN*-ul este în starea 1, iar caracterul curent de la intrare este primul caracter al șirului. Se execută tranziția (fără a se citi caracterul curent de la intrare) și se ajunge astfel în starea 2. Pentru caracterul de intrare `'a'`, executăm tranziția cu `'a'`, ajungând tot în starea 2. Caracterul curent de intrare devine cel de-al doilea caracter, `'b'`. Se execută tranziția cu `'b'` și se ajunge în starea finală 3, deci șirul `"ab"` este acceptat de *AFN*.

Analog se poate vedea că șirurile `"b"`, `"aab"`, `"aaab"` sunt acceptate de *AFN*, în timp ce șirurile `"a"`, `"ac"`, `"acb"` nu sunt acceptate. De fapt se observă că limbajul șirurilor acceptate de *AFN* este exact limbajul descris de expresia regulată `"a*b"`.

Cum se construiește un AFN dintr-o expresie regulată?

O expresie regulată poate avea mai multe *AFN*-uri echivalente. *AFN*-urile respective pot avea un număr diferit de stări și structuri distincte, dar execuțiile lor trebuie să fie echivalente. Există un algoritm (algoritmul de construcție al lui ***Thompson***) care, pe baza arborelui de derivare generat din expresia regulată, poate construi un *AFN* cu următoarele proprietăți adiționale:

- *AFN*-ul are o singură stare finală;
- fiecare stare a *AFN*-ului are:
  - fie o singură tranziție non-ε și nimic altceva;
  - fie cel mult două tranziții ε și nimic altceva, caz în care se numește ***null-state***.

Pentru a transforma (mai științific, ***a compila***) o expresie regulată într-un *AFN* trebuie să apelăm, după cum am spus, la arborele de derivare. Pentru aceasta trebuie realizată analiza sintactică a expresiei regulate. Blocul care realizează această operație este denumit în teoria compilatoarelor ***analizor sintactic*** și este blocul care urmează după analizorul lexical. Funcția analizorului sintactic (a *parser*-ului) este de a determina dacă șirul de caractere de la intrare este o expresie regulată și de a construi arborele de derivare corespunzător.

Algoritmul lui Thompson...

...pentru construcția unui AFN dintr-o expresie regulată

Intrare

Se consideră o expresie regulată r peste alfabetul de caractere acceptate Σ.

Ieșire

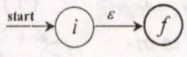
AFN-ul corespunzător expresiei.

Metoda

Mai întâi se va parsa expresia în subexpresiile sale, apoi utilizând regulile (1) și (2), pe care le vom prezenta în continuare, se vor construi *AFN*-uri pentru fiecare simbol de bază (acesta poate fie un simbol din alfabet, fie ε). Este important de observat că, dacă un simbol apare de mai multe ori în r, atunci va fi construit câte un *AFN* pentru fiecare apariție a sa.

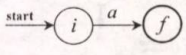
Apoi, în funcție de structura sintactică a expresiei regulate se vor combina aceste *AFN*-uri, folosind regula (3), până când se va obține *AFN*-ul pentru întreaga expresie.

(1). Pentru ε construim *AFN*-ul:



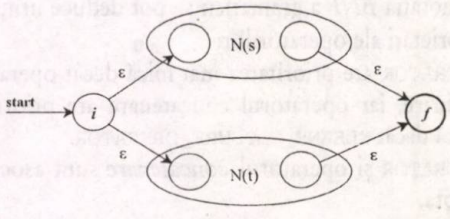
unde i este starea inițială, iar f este starea finală.

(2) Pentru a din Σ construim *AFN*-ul:

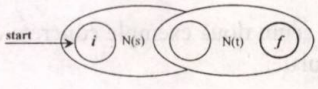


(3) Presupunem că $N(s)$ și $N(t)$ sunt *AFN*-urile pentru expresiile s și t.

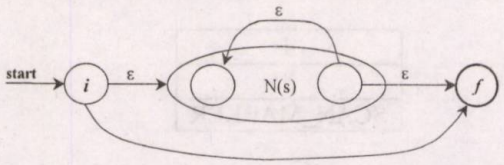
a) Pentru expresia regulată `s|t`, se va construi următorul *AFN*:



b) Pentru expresia regulată `st`, se va construi *AFN*-ul:



c) Pentru expresia regulată `s*` vom avea:



d) În cazul în care expresia regulată este în paranteză `(s)` vom construi *AFN*-ul pentru *N(s)* și vom ignora parantezele.

Se observă că algoritmul introduce cel mult două stări în *AFN* pentru fiecare caracter din expresia regulată. Așadar, putem afirma că *AFN*-ul are cel mult de două ori mai multe stări decât numărul de caractere din care este construită expresia regulată.

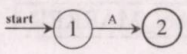
Exemplu

Vom construi *AFN*-ul pentru expresia regulată

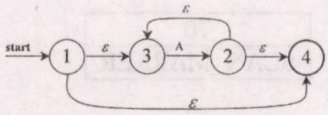
```
r = A*B(C|D),
```

al cărei arbore de derivare a fost prezentat anterior.

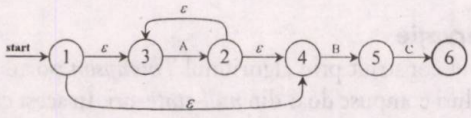
Pentru subexpresia `"A"` (primul caracter din expresie), aplicând regula (2) obținem *AFN*-ul:



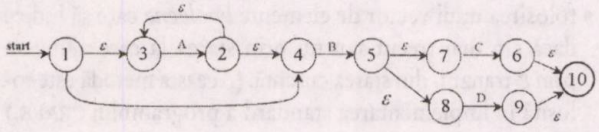
La pasul următor întâlnim operatorul `'*'`, aplicăm regula (3.c) și obținem:



Aplicăm regula (2) când întâlnim subexpresia `"B"`, iar când întâlnim subexpresia `"(C|D)"`, memorăm faptul că starea 5 este starea de început a acestei subexpresii și, inițial, prelucrăm doar subexpresia `"C"`:



Știm că starea 5 este starea de început a subexpresiei `"(C|D)"`. Când întâlnim operatorul `'|'`, aplicăm regula (3.a) și obținem forma finală a *AFN*-ului:



Se impun câteva explicații:

- etapele creării *AFN*-ului prezentate mai sus sunt exact etapele pe care le parcurge și programul `GRAPE` când convertește o expresie regulată în *AFN*.
- s-ar putea să vă mirați de faptul că o stare creată la un anumit moment își poate schimba numărul de ordine la un pas ulterior. Așa este concepută implementarea algoritmului lui *Thompson* în `GRAPE`. Puteți studia sursa programului `GRAPE`, ea fiind cea mai grăitoare descriere a implementării.

Prezentarea analizorului sintactic

Am spus că algoritmul de construcție a *AFN*-ului din expresia regulată apelează la *parser* (analizor sintactic). Metoda de parsare care este folosită de programul `GRAPE` este denumită ***top-down parsing*** (sau ***recursive-descent***). Ea funcționează în felul următor: pentru o expresie, arborele de derivare corespunzător se construiește pornind de la rădăcină până când se ajunge la frunze (de aici provine denumirea *top-down*).

Metoda se implementează utilizând descrierea *BNF* a gramaticii expresiilor regulate; mai exact, realizăm o "traducere" în limbaj de programare a fiecărei producții a gramaticii. Așadar, analizorul este compus din următoarele funcții: `Expression`, `Factor` și `Term`.

Aceste funcții conțin, pe lângă "traducerile" producțiilor corespunzătoare, și acțiunile de construire a *AFN*-ului (științific, aceste acțiuni sunt numite ***acțiuni semantice***).

Cum este reprezentat intern un AFN?

Proprietățile *AFN*-ului construit cu ajutorul algoritmului lui *Thompson* ne determină să folosim următoarea reprezentare pentru acesta:

- un vector `ch`, unde elementul arbitrar `ch[i]` reprezintă caracterul asociat tranziției care pornește din starea numerotată cu *i*. Acest caracter poate să fie unul acceptat de *AFN*, sau poate fi `NULL_STATE_CHAR` (ceea ce semnifică faptul că starea respectivă este *null-state*);
- doi vectori `next1` și `next2`, unde `next1[i]` și `next2[i]` reprezintă stările în care sosesc tranzițiile care pornesc din starea *i*. Vectorii pot avea valori `INVALID_TRANSITION`, ceea ce înseamnă că nu există tranziția respectivă.
- cei trei vectori sunt statici și au aceeași dimensiune: dublul lungimii maxime acceptate a șirului expresiei regulate.

Cum se execută AFN-ul?

După construirea *AFN*-ului, putem să verificăm dacă respectivul automat acceptă anumite șiruri de caractere. Aceasta se realizează cu funcția:

```
int Match(char *text),
```

unde `text` este șirul de caractere *ASCII* care va fi sau nu acceptat de către *AFN*, iar valoarea returnată de funcție reprezintă numărul de caractere conținute de subșirul prefix de lungime maximă din text, care se potrivește peste expresia regulată echivalentă cu *AFN*-ul (dacă rezultatul este un întreg strict pozitiv).

Implementarea acestei funcții folosește structura de date ***DEQue*** (*Double-Ended Queue*), o structură la care se pot adăuga elemente în capul (funcția `AddHead()`) și coada structurii (funcția `AddTail()`), dar din care elementele se extrag doar de la cap (funcția `GetHead()`). Această structură combină proprietățile unei cozi cu cele ale unei stive.

În *DEQue* există, în orice moment, două tipuri de stări: unele sunt depozitate în "jumătatea" superioară a structurii (care conține capul structurii), iar celelalte în "jumătatea" inferioară.

Pentru a se delimita cele două "jumătăți" (care pot fi inegale) se folosește un element special, denumit `SCAN_MARKER`, care nu poate apărea în nici una dintre cele două "jumătăți".

În "jumătatea" superioară se memorează stările în care se poate afla *AFN*-ul înainte de a potrivi caracterul curent din șirul de intrare, iar în "jumătatea" inferioară stările fii ale stărilor din prima "jumătate" care au potrivit caracterul curent din șirul de intrare.

Observație

Execuția *AFN*-ului este un caz particular de parcurgere în lățime.

Exemplu

Vom descrie acum modalitatea de funcționare a funcției `Match()`, exemplificând pe *AFN*-ul construit la exemplul anterior și șirul de intrare `AABDE`.

Inițial, trebuie potrivit primul caracter din text, iar în *DEQue* avem:

1
SCAN_MARKER

Extragem `1` din coadă. Observăm că din starea `1` se poate ajunge în stările `3` și `4` prin tranziții ε, așadar *DEQue* va arăta astfel:

4
3
SCAN_MARKER

Extragem `4` din coadă: starea `4` are o singură tranziție cu caracter `'B'` asociat, deci nu putem să folosim starea `4`. Extragem starea `3` din coadă: starea `3` are o tranziție cu caracter `'A'` asociat, deci executăm `AddTail()` cu starea fiu a stării `3`. *DEQue* va fi:

SCAN_MARKER
2

Ajungem să extragem din coadă elementul `SCAN_MARKER`; vom înainta la următorul caracter din text și vom insera un element `SCAN_MARKER` în coadă:

2
SCAN_MARKER

Din starea `2` se poate ajunge în stările `3` și `4` prin tranziții ε, așadar *DEQue* va fi:

4
3
SCAN_MARKER

Din nou potrivim cu tranziția spre starea `2` cel de-al doilea caracter `'A'` din text.

SCAN_MARKER
2

Întâlnim `SCAN_MARKER`; înaintăm la al treilea caracter din text, `'B'`. *DEQue* va fi:

2
SCAN_MARKER

După aceea *DEQue* va fi din nou:

4
3
SCAN_MARKER

Caracterul curent din text este potrivit doar de starea `4`, așa că înaintăm la caracterul `'D'` din text. *DEQue* va fi:

5
SCAN_MARKER

Din starea `5` se poate ajunge prin tranziții ε la stările `7` și `8`:

8
7
SCAN_MARKER

Caracterul curent din text este potrivit doar de starea `8`; înaintăm la caracterul `'E'` din text. *DEQue* va fi:

9
SCAN_MARKER

Din starea `9` se ajunge prin ε-tranziție în starea `10`. *DEQue* va fi:

10
SCAN_MARKER

În momentul în care extragem starea `10` din coadă, observăm că `10` este stare finală. Așadar șirul `"AABD"` este acceptat de *AFN*. Drept urmare, funcția va returna valoarea `4` (lungimea maximă a șirului prefix al textului de intrare care este acceptat de *AFN*).

Observație

AFN-ul construit prin algoritmul *Thompson* poate conține cicluri compuse doar din *null-state*-uri. În acest caz, este posibil ca, atunci când calculăm stările în care se poate ajunge prin ε-tranziții dintr-o stare, să intrăm într-o buclă infinită, care să ducă la umplerea până la refuz a structurii *DEQue*. Există două soluții pentru această problemă:

- folosirea unui vector de elemente *booleene* care să indice dacă am mai trecut sau nu prin starea la care ajungem prin ε-tranziții din starea curentă. (Aceasta metodă este folosită în implementarea standard a programului `GRAPE`.)
- Mulțimile de stări în care se poate ajunge din orice stare prin ε-tranziții (implementarea *eps_closure*) se pot determina înainte de apelul funcției `Match()`.

Pentru a găsi subșirul (nu neapărat prefix) de lugime maximă, acceptat de *AFN*, se vor executa apeluri succesive de tipul `Match(text)`, `Match(text+1)`, ..., `Match(text+strlen(text)-1)`, memorându-se poziția și lungimea prefixului maximal găsit de toate funcțiile `Match()` apelate.

Analiza complexității programului

Timpul de execuție al *parser*-ului are un ordin de complexitate proporțional cu L, unde L este numărul de caractere al expresiei regulate.

Timpul de execuție al funcției `Match()` are un ordin de complexitate $O(M \cdot N)$, unde M este numărul de stări din care este compus *AFN*-ul, iar N este numărul de caractere ale textului în care se încearcă potrivirea.

Necesarul de memorie are ordinul de complexitate $O(M)$.

Alte utilizări ale expresiilor regulate

Expresiile regulate au fost studiate pentru prima dată de către *Kleene* în anul 1956; el era interesat în a descrie evenimentele reprezentate de automatele finite care modelează activitatea neuronală.

Datorită faptului că sunt foarte descriptive:

- se pot folosi în găsirea cuvintelor scrise greșit; de exemplu expresia regulată `".*(ie+ei).*"` găsește cuvintele care conțin `"ie"` sau `"ei"`; aceste cuvinte au o probabilitate relativ mare de a fi scrise greșit în limba engleză.
- pot determina dacă o adresă *e-mail* este scrisă corect sau nu.
- se folosesc la analiza *script*-urilor server *Apache*.
- se folosesc în interogările *SQL*.

Jarvis (în anul 1976) a folosit expresiile regulate pentru a găsi imperfecțiuni în cablajele imprimate.

O mulțime de programe au inclus suport pentru expresii regulate: *SED*, *AWK*, *PERL*, *PHP*, *VIM*, *LEX*, *FLEX*.

La final...

Fișierele sursă ale pachetului GRAPE sunt:

- **`COMDEF.H`** - definiții de tipuri de date;
- **`DEQUE.H`** - clasa DEQue (*Double Ended Queue*)
- **`NFA.CPP`** și **`NFA.H`** - clasa AFN (*Nondeterministic Finite Automata*) - conține funcțiile de analiză lexicală și de potrivire;
- **`GRAPE.CPP`** - interfața programului cu utilizatorul; se citesc expresiile regulate din fișierul **`REGEXP.IN`** și textul din fișierul **`TEXT.IN`**, se descrie fiecare *AFN* asociat fiecărei expresii regulate în fișierul **`NFA_DIAG.OUT`** în fișierul **`REGEXP.OUT`** se scriu detalii despre construcția și execuția fiecărui *AFN* pentru textele de intrare.

Notă

Programul GRAPE a fost testat de autori și de redacția *GInfo*. Totuși, nici autorul, nici membrii redacției nu pot fi făcuți responsabili pentru eventuale daune provocate de funcționarea incorectă a programului.

Observație

E interesant de remarcat că GRAPE este, de fapt, un compilator, compus din:

- un modul de analiză lexicală de o complexitate foarte redusă (care verifică dacă s-a citit sau nu un caracter alfanumeric, dacă `'\'` este caracterul curent etc.; în cazul apariției caracterului `'\'` se trece la caracterul următor prin efectuarea unei citiri suplimentare);
- un modul de analiză sintactică în cadrul căruia se realizează și traducerea expresiei în *AFN*;
- un modul de *matching* (potrivire) despre care se poate spune că realizează interpretarea *AFN*-ului, având ca date de intrare textul.

Trebuie remarcat faptul că, în programul GRAPE, modulul de analiză lexicală este îmbinat cu cel de analiză sintactică.

Alte implementări

- **ftp://ftp.zoo.toronto.edu/pub/bookregexp.{tar|shar}** - implementarea unui *parser* de expresii regulate cu *AFN*.
- *GREP* (*Get Regular Expression Parser*)
- în numărul din decembrie 2000 al revistei *Dr. Dobbs Journal*, a apărut un articol cu titlul "*How Can I Extend Java's Search Capabilities?*", care conține și sursele *Java* ale unui utilitar de tip *GREP*. Acestea au fost publicate online și sunt disponibile în format electronic pe *site*-ul **www.ddj.com**.

Programul GREP

Merită prezentată o scurtă istorie a programului *GREP*: prima versiune a fost scrisă de ***Ken Thompson*** și era o implementare de parser cu *AFN* (ca și GRAPE).

În 1976, ***Al Aho*** a realizat o nouă implementare a acestui program (pe care a denumit-o ***EGREP*** - *Extended GREP*) și care folosește un automat finit determinist. *EGREP* avea următoarele caracteristici: analiza sintactică dura mai mult decât în cazul primei versiuni de *GREP*, dar partea de execuție a automatului finit era mai rapidă. Ca urmare, ***Aho*** a adus o îmbunătățire ulterioară *EGREP*-ului - o tehnică pe care a numit-o "*cached lazy evaluation*", care surclasează celelalte versiuni de *GREP*. Implementarea *GNU GREP* are surse publice.

Bibliografie

[1] Robert Sedgewick, *Algorithms*, Addison-Wesley, 1984 - capitolele *Pattern Matching* și *Parsing*

[2] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Compilers. Principles, Techniques and Tools*, Addison-Wesley, 1986

[3] Mihai Budiu, *Expresii regulate*, PC-Report, Computer Press Agora, martie 2000

[4] Alexander Pereira Calsavara, *How Can I Extend Java's Search Capabilities?*, Dr. Dobbs Journal, decembrie 2000

[5] Mioara Avram, *Gramatica pentru toți*, Editura Academiei RSR, 1986

Alexandru Șușu este student în anul IV la Facultatea de Automatică și Calculatoare din cadrul Universității Politehnica din București. Poate fi contactat prin e-mail la adresa **asusu@home.ro**.

Bursele AGORA (Ediția 2001/2002)

În acest număr sunt publicate enunțurile problemelor de la runda a treia. Dacă nu v-ați înscris încă, puteți participa începând cu această rundă. Pentru informații suplimentare puteți vizita site-ul oficial al concursului la adresa www.ginfo.ro/concurs. Aici veți putea găsi enunțurile problemelor, clasamente parțiale, soluții oficiale, date de test folosite, anunțuri generale etc.

Participare

Pentru a participa la concursul *Bursele Agora* începând cu runda a treia, vă rugăm să completați integral talonul destinat acestui scop și să îl trimiteți până la data de *15 ianuarie 2002* (data poștei). Taloanele trimise după această dată vor da dreptul de participare doar începând cu a patra rundă. Taloanele incomplete nu vor fi luate în considerare. Redacția *GInfo* nu își asumă nici o răspundere pentru taloanele care, din diferite motive, nu ajung în timp util la redacție.

În termen de *cinci* zile de la primirea talonului de către redacție, vă vom trimite prin *e-mail* codul dumneavoastră de identificare.

Concurenții care au primit deja codul de identificare nu trebuie să trimită și acest talon.

Vă rugăm să citiți cu atenție noul regulament. Există câteva modificări minore care constau, în special, în modalitatea de trimitere a soluțiilor și cea de înscriere.

Runda a treia

Cele trei probleme ale rundei a treia sunt *Templu*, *Scânduri* și *Cub*.

Prima problemă a fost propusă de către *Mihai Scorțaru*, student în anul IV la *Universitatea Tehnică* din Cluj, redactor-șef adjunct al revistei noastre și organizator principal al concursului.

Cea de-a doua problemă a fost propusă de *Mihai Stroe*, student în anul IV la *Universitatea Politehnica* din București, fost olimpic internațional, câștigător al primei ediții a concursului nostru și clasat pe locul al doilea la ediția de anul trecut.

Ultima problemă a fost propusă de către *Claudiu Soroiu*, student în anul II la *Universitatea Babeș-Bolyai* din Cluj și, începând cu acest număr, redactor *GInfo*.

Rezolvările celor trei probleme pot fi trimise până pe data de *31 ianuarie 2002*. Rezultatele rundei vor deveni publice pe data de *15 februarie 2002*.

Pentru a evita întârzierile, vă sfătuim să trimiteți soluțiile cu câteva ore sau zile înaintea expirării termenului limită de trimitere a acestora.

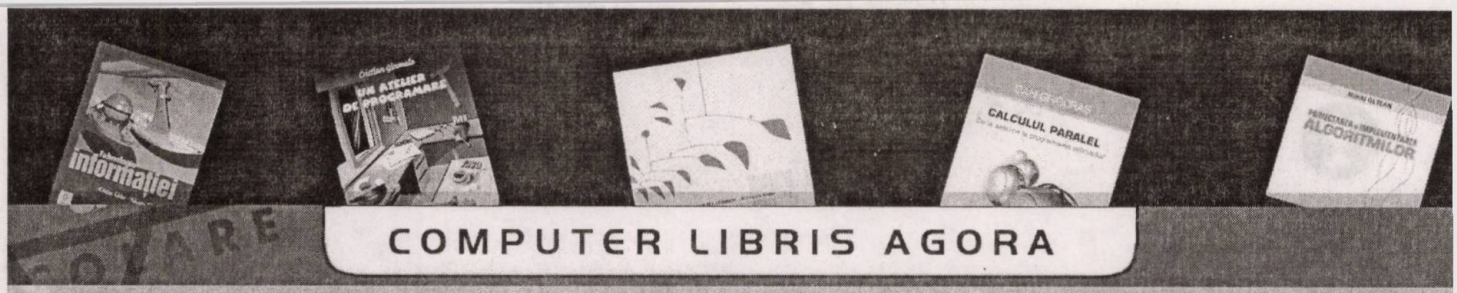
Restricții și precizări

Nu se vor accepta rezolvări care ajung la redacție după termenul limită specificat, chiar dacă acesta este depășit doar cu câteva minute.

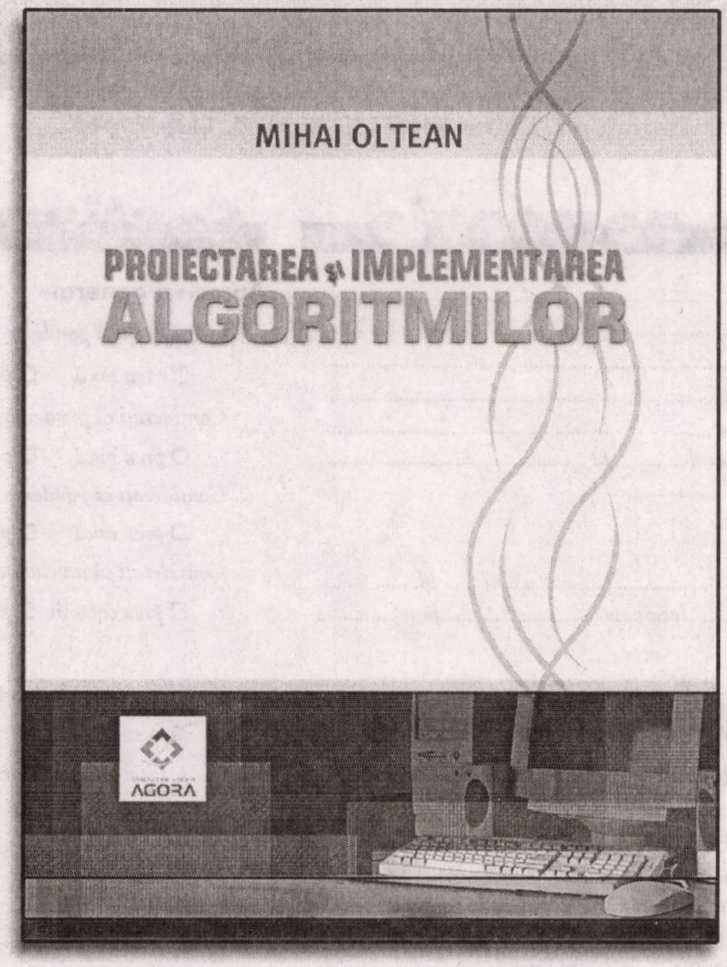
Redacția nu va trimite prin *e-mail* enunțurile problemelor, indiferent de motivul pentru care un concurent ar putea solicita acest lucru.

La mulți ani!

Constantin Comărniceanu, București - 1 XII 1983
Kanti-Sorin Arion, Râmnicu Sărat - 1 XII 1984
Kovács Boglárka, Sfântu Gheorghe - 3 XII 1985
Adrian Airinei, Bacău - 7 XII 1984
Mihai Cosma, Bistrița - 8 XII 1983
Perini Alpár-Ferenc, Sfântu Gheorghe - 11 XII 1984
Daniel Spighel, Cluj-Napoca - 13 XII 1980
Radu Naidinescu, București - 13 XII 1981
Aurelian Ghiță, Buzău - 19 XII 1983
Alexandru Moșoi, Bacău - 19 XII 1984
Simona Cană, Vaslui - 23 XII 1984
Mihai Maerean, București - 24 XII 1980
Victor Asavei, Drobeta Turnu-Severin - 25 XII 1981
Alexandru Crivăț, Pitești - 26 XII 1985
Cosmin Lehene, Zalău - 27 XII 1983
Vlad Vanca, Satu Mare - 27 XII 1983
Victor-Marius Costan, București - 29 XII 1983
Sergiu Spătaru, Zimandu Nou - 31 XII 1985



COMPUTER LIBRIS AGORA



"Cum aș putea să justific existența acestei cărți? Cum aș putea spre exemplu să justfic cele 100 de pagini despre programarea dinamică, acum când această metodă nu mai figurează în programele școlare? Nu știu. Sau, cum aș putea să justific capitolul despre algoritmi genetici, când pe plan mondial abia de câțiva ani a fost dovedită utilitatea și eficiența lor? Din nou nu știu! Dar rămâne bănuiala: a scrie algoritmi înseamnă a crea. Și nu poți crea fără să gândești."

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

COMPUTER LIBRIS AGORA

Rundele 3, 4, 5!

Talon de PARTICIPARE

Acest talon dă dreptul de participare la Concursul de Programare Agora începând cu a treia rundă. Completați integral acest talon și trimiteți-l pe adresa redacției: str. Universității nr. 8 ap. 8, Cluj-Napoca.

Date personale

Nume: .....
Prenume: .....
Telefon: .....
Adresă e-mail: .....
❒ *elev în clasa* ..... *la* .....
❒ *student în anul* ..... *la* .....
Data nașterii: .....

Adresa

str. ..... *nr.* .... *bl.* ..... *sc.* .....
apt. ..... *cod* ..... *localitate* ..... *județ* .....

Chestionar

Folosiți un calculator acasă? ❒ *Da* ❒ *Nu*
Aveți acces e-mail acasă? ❒ *Da* ❒ *Nu*
Aveți acces WWW acasă? ❒ *Da* ❒ *Nu*
Plănuiți un upgrade anul acesta? ❒ *Da* ❒ *Nu*
Cât timp petreceți în fața calculatorului? ..... *ore/săptămână*
Cât timp petreceți online? ..... *ore/săptămână*
Cât timp vă jucați? ..... *ore/săptămână*
Cât timp programați? ..... *ore/săptămână*
Câte calculatoare sunt la școală/facultate? ..... *buc.*
Cum apreciați dotarea cu tehnică de calcul a școlii?
❒ *Dezastruoasă* ❒ *Foarte slabă* ❒ *Slabă*
❒ *Nesatisfăcătoare* ❒ *Medie* ❒ *Satisfăcătoare*
❒ *Bună* ❒ *Foarte bună* ❒ *Excelentă*
Considerați că prețul revistei este:
❒ *prea mic* ❒ *accesibil* ❒ *prea mare*
Ați dori ca revista să conțină și un CD? ❒ *Da* ❒ *Nu*
Ați dori ca revista să fie color? ❒ *Da* ❒ *Nu*
De cât timp cunoașteți GInfo? ..... *ani*
Cum ați procurat acest număr?
❒ *chioșc* ❒ *abonament* ❒ *cadou* ❒ *altfel:* .....
Ați dori ca GInfo să aibă:
❒ *4 numere/an* ❒ *8 numere/an*
❒ *12 numere/an* ❒ *24 numere/an*

Conținutul GInfo

Aprecieri generale

Considerați că ponderea problemelor rezolvate este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că ponderea problemelor propuse este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că ponderea subiectelor teoretice este:
❒ *prea mică* ❒ *potrivită* ❒ *prea mare*
Considerați că nivelul de accesibilitate al articolelor este:
❒ *prea coborât* ❒ *potrivit* ❒ *prea ridicat*

Acordați note de la 1 la 10 pe următoarele criterii:
Structura tematică: .....
Calitatea articolelor: .....
Prezentare grafică: .....
Calitate tipografică: .....

În viitor

În următoarele numere așteptați:
mai multe articole de programare? ❒ *Da* ❒ *Nu*
mai multe articole teoretice? ❒ *Da* ❒ *Nu*
mai multe articole despre Internet? ❒ *Da* ❒ *Nu*
mai multe prezentări de programe? ❒ *Da* ❒ *Nu*
prezentări de jocuri? ❒ *Da* ❒ *Nu*
articole despre hardware? ❒ *Da* ❒ *Nu*

V-ar plăcea să citiți:
pagini redactate de elevii unor licee? ❒ *Da* ❒ *Nu*
lucrări premiate, realizate de elevi? ❒ *Da* ❒ *Nu*
actualități școlare? ❒ *Da* ❒ *Nu*
prezentarea unor situri Web? ❒ *Da* ❒ *Nu*
tips & tricks? ❒ *Da* ❒ *Nu*

Calculatorul

Indicați caracteristicile calculatorului pe care îl veți folosi la rezolvarea problemelor de la concurs.

Tipul procesorului

- ❒ *Intel Pentium*
- ❒ *Intel Pentium Pro*
- ❒ *Intel Pentium II*
- ❒ *Intel Pentium III*
- ❒ *Intel Pentium 4*
- ❒ *Intel Celeron*
- ❒ *AMD K6*
- ❒ *AMD K6-2*
- ❒ *AMD K6-III*
- ❒ *AMD Athlon*
- ❒ *AMD Duron*
- ❒ *Altul:* .....

Frecvența procesorului : ..... MHz

Tipul plăcii de bază: .....

Frecvența magistralei sistem: ..... MHz

Capacitatea memoriei sistem: ..... MB

Tipul plăcii video: .....

Capacitatea memoriei plăcii video: ..... MB

Tipul discului hard: .....

Capacitatea discului hard: ..... GB

Componente suplimentare

Unitate CD-ROM: ❒ Da ❒ Nu
Tip: .....
Viteza de citire: ..... ×

Unitate CD-RW: ❒ Da ❒ Nu
Tip: .....
Viteza de citire: ..... ×
Viteza de inscripționare: ..... ×
Viteza de reinscripționare: ..... ×

Placă de sunet: ❒ Da ❒ Nu
Tip: .....

Boxe active: ❒ Da ❒ Nu
Tip: .....

Placă de rețea: ❒ Da ❒ Nu
Tip: .....

Modem: ❒ Da ❒ Nu
Tip: .....
Viteza de transfer: ..... Kbps

Sistemul de operare

- ❒ *MS-DOS*
- ❒ *Microsoft Windows '95 OSR 2*
- ❒ *Microsoft Windows '98 Second Edition*
- ❒ *Microsoft Windows NT 4.0*
- ❒ *Microsoft Windows XP*
- ❒ *Microsoft Windows '95*
- ❒ *Microsoft Windows '98*
- ❒ *Microsoft Windows Me (Millennium Edition)*
- ❒ *Microsoft Windows 2000*
- ❒ *Altul:* .....

Aplicații folosite frecvent

Marcați aplicațiile instalate pe calculator și indicați versiunile corespunzătoare

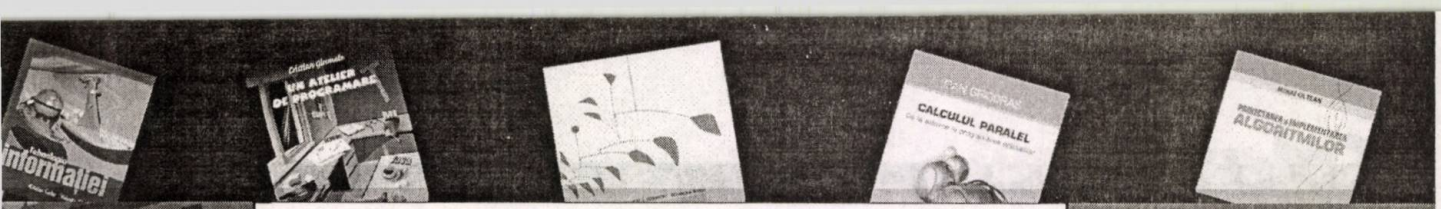
- ❒ *Microsoft Internet Explorer (versiunea .....)*
- ❒ *Microsoft Office (versiunea .....)*
- ❒ *Microsoft Visual Studio (versiunea .....)*
- ❒ *Borland Pascal (versiunea .....)*
- ❒ *Corel Draw (versiunea .....)*
- ❒ *Borland C++Builder (versiunea .....)*
- ❒ *Netscape Navigator (versiunea .....)*
- ❒ *Adobe Photoshop (versiunea .....)*
- ❒ *Adobe Acrobat Reader (versiunea .....)*
- ❒ *Borland C++ (versiunea .....)*
- ❒ *Borland Delphi (versiunea .....)*
- ❒ *Borland JBuilder (versiunea .....)*

Programe antivirus utilizate: .....

Arhivatoare utilizate: .....

Jocurile preferate: .....

Site-urile preferate: .....

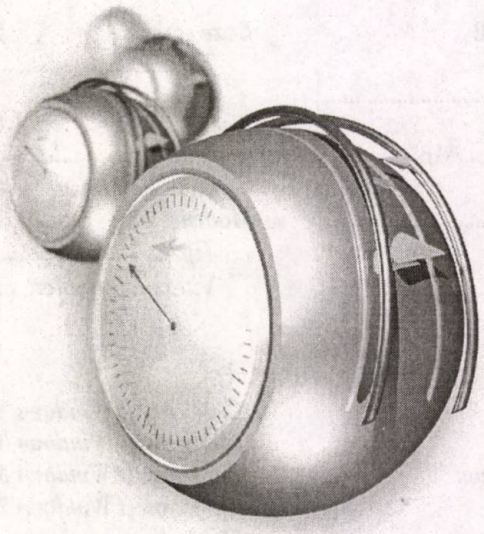


COMPUTER LIBRIS AGORA

DAN GRIGORAȘ

CALCULUL PARALEL

De la sisteme la programarea aplicațiilor



Calculul paralel. De la sisteme la aplicații tratează, într-un mod natural și structurat, cele mai importante aspecte ale proiectării aplicațiilor paralele, sau, cu un termen mai nou, de mare performanță. Paralelismul, care înseamnă executarea mai multor operații simultan, este o caracteristică a tuturor calculatoarelor actuale, inclusiv a celor personale. De aceea, cunoașterea principiilor și legilor care îl guvernează, ca și a metodologiilor pentru proiectarea aplicațiilor cele mai performante este o necesitate,dacă se dorește exploatarea optimă a resurselor sistemelor de calcul. Parametrii de performanță și metodologiile de proiectare și execuție optimă reprezintă subiectele care guvernează discursul științific al cărții de față.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Runda 3, problema 1

TEMPLU

Denumirea subdirectorului: **TEMPLE**

Denumirea fișierului de intrare: **TEMPLE.IN**

Denumirea fișierului de ieșire: **TEMPLE.OUT**

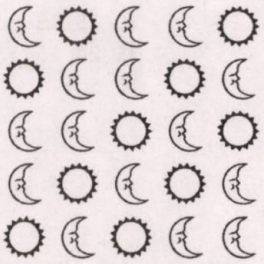
Descrierea problemei

Arheologii au descoperit ruinele unui vechi templu. Pe un perete se află o inscripție foarte interesantă, formată din simboluri ale lunii și ale soarelui. Inscripția apare sub forma unei matrice pătratice cu N linii și N coloane.

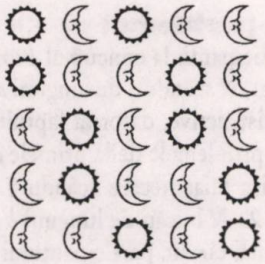
Din nefericire, trecerea timpului a dus la deteriorarea peretelui și acum numai ultima coloană mai este vizibilă. Până la urmă, arheologii au reușit să descopere o regulă interesantă chiar dacă simbolurile nu mai sunt integral vizibile.

Se consideră simbolurile lunii și ale soarelui de pe o linie care sunt dispuse într-o ordine aparent aleatoare. Totuși, în continuare se observă o regulă foarte clară. O altă linie a matricei se obține din prima, efectuând o permutare cu o poziție, adică primul element ajunge pe ultima coloană, toate celelalte sunt mutate la stânga cu o poziție. O altă linie se obține permutând din nou cu o poziție și așa mai departe, până la obținerea tuturor rândurilor.

De exemplu, în situația ipotetică în care un rând ar fi atunci cele cinci rânduri ale inscripției sunt:



Este evident că toate aceste rânduri apar pe inscripție, dar ele nu se află în această ordine. S-a observat că există o regulă de ordonare lexicografică a acestora. Singura regulă care stă la baza acestei ordonări este aceea că soarele se află, din punct de vedere lexicografic, înaintea lunii. Așadar, fiecare rând poate fi privit ca un cuvânt și aceste cuvinte sunt ordonate folosind această regulă. În acest caz, pentru linia considerată, configurația inscripției ar fi:



Folosind regulile prezentate, va trebui să determinați prima linie a inscripției (celelalte sunt ușor de aflat dacă este cunoscută aceasta), având în vedere faptul că aveți la dispoziție doar simbolurile de pe ultima coloană.

Date de intrare

Prima linie a fișierului de intrare va conține un număr întreg N, care va indica dimensiunea inscripției. Următoarea linie va conține N numere întregi, separate prin câte un spațiu, care vor indica simbolurile de pe ultima coloană a inscripției. Pentru a simboliza luna se va folosi valoarea `1`, iar pentru a simboliza soarele se va folosi valoarea `0`.

Date de ieșire

Fișierul de ieșire va conține o singură linie pe care se vor afla N numere întregi, separate prin câte un spațiu, care vor reprezenta simbolurile de pe prima linie a inscripției. Se va folosi valoarea `0` pentru soare și valoarea `1` pentru lună.

Restricții

- $1 \leq N \leq 50000$;
- Datele de test sunt astfel alese încât să permită determinarea soluției.

Exemplu

TEMPLE.IN	TEMPLE.OUT
5 1 1 1 0 0	0 1 0 1 1

Timp de execuție/test: **1 secundă**

Runda 3, problema 2

SCÂNDURI

Denumirea subdirectorului: **PLANKS**

Denumirea fișierului de intrare: **PLANKS.IN**

Denumirea fișierului de ieșire: **PLANKS.OUT**

Descrierea problemei

Unul dintre participanții la concursul *Bursele Agora* avea, la un moment dat, N scânduri de lungimi $a_1, a_2, ..., a_N$. Mânat de porniri distructive, datorită faptului că nu a reușit să rezolve toate problemele de la primele două runde, el a luat un topor și a tăiat fiecare scândură în două bucăți. Astfel a obținut $2 \cdot N$ bucăți de lungimi $b_1, b_2, ..., b_{2N}$.

Câteva zile mai târziu, participantului i-a părut rău că a stricat scândurile și a dorit să reconstituie scândurile inițiale. Din nefericire, bucățile s-au amestecat, deci el nu a mai știut cum să le lipească.

Sarcina voastră este de a scrie un program care determină perechile de bucăți care trebuie lipite pentru a reconstitui scândurile inițiale.

Date de intrare

Prima linie a fișierului de intrare conține numărul N care reprezintă numărul scândurilor inițiale. Pe următoarele N linii se află N numere întregi, reprezentând lungimile celor N scânduri inițiale. Pe următoarele $2 \cdot N$ linii se află $2 \cdot N$ numere întregi care reprezintă cele $2 \cdot N$ lungimi ale bucăților obținute după tăiere.

Date de ieșire

Pe prima linia a fișierului de ieșire se află numărul K al scândurilor reconstituite. Pe următoarele K linii se află câte trei numere întregi S, X și Y ($S = X + Y$) cu semnificația: o scândură de lungime S este reconstituită din două bucăți de lungime X și Y.

Numerele S reprezintă lungimi ale scândurilor inițiale. Dacă o valoare S apare la începutul unor linii din fișierul de ieșire de x ori, atunci trebuie să existe cel puțin x scânduri de lungime S. Numerele X și Y reprezintă lungimi ale bucăților obținute după tăiere.

Dacă o valoare X apare ca al doilea sau al treilea număr pe o linie a fișierului de ieșire de y ori, atunci trebuie să existe cel puțin y bucăți obținute după tăiere care au lungimea X.

Restricții și precizări

- $1 \le N, b_i \le 100$;
- nu există mai mult de cinci scânduri care să aibă aceeași lungime;
- nu există mai mult de cinci bucăți obținute după tăiere care să aibă aceeași lungime;
- datele de intrare sunt alese în așa fel încât să se poată reconstitui toate cele N scânduri ($K = N$);
- în cazul în care sunt reconstituite toate cele N scânduri, veți primi întregul punctaj alocat unui anumit test.
- în cazul în care nu sunt reconstituite toate cele N scânduri, dar sunt reconstituite cel puțin $[3 \cdot N / 4]$ dintre acestea, veți primi $[P / 2]$ din cele P puncte alocate unui anumit test.

Exemplu

PLANKS.IN	PLANKS.OUT
6	6
10	15 10 5
15	20 10 10
20	25 10 15
25	30 15 15
30	35 20 15
35	10 5 5
5	
5	
5	
10	
10	
10	
10	
15	
15	
15	
15	
20	

Timp de execuție/test: **1 secundă**

CUB

Denumirea subdirectorului: **CUBE**

Denumirea fișierului de intrare: **CUBE.IN**

Denumirea fișierului de ieșire: **CUBE.OUT**

Descrierea problemei

Se consideră un cub de dimensiune N. Acesta poate fi împărțit în N^3 cubulețe cu latura egală cu unitatea. În fiecare dintre aceste cubulețe se află câte o literă mică a alfabetului latin.

Sarcina voastră este de a determina numărul de apariții ale unui cuvânt S de lungime K, format din litere mici distincte. Cuvântul apare în cub dacă prima sa literă apare într-un anumit cubuleț, cea de-a doua literă apare în unul dintre cele cel mult 26 de cubulețe învecinate, a treia literă apare în unul dintre cele cel mult 26 de cubulețe vecine cu cubulețul în care se află a doua literă și așa mai departe.

Pentru un cubuleț de coordonate (x, y, z), cele 26 de cubulețe învecinate au coordonatele $(x + d_x, y + d_y, z + d_z)$, unde d_x, d_y și d_z fac parte din mulțimea $\{-1, 0, 1\}$ și $|d_x| + |d_y| + |d_z| > 0$.

Aparițiile cuvântului se pot suprapune parțial, în sensul că un anumit cubuleț poate apărea în două sau mai multe apariții ale cuvântului dat.

Date de intrare

Pe prima linie a fișierului de intrare se află cuvântul S. Pe a doua linie se află un număr întreg N care reprezintă dimensiunea cubului.

Pe următoarele N^2 linii se află câte N litere ale alfabetului latin neseparate prin spații.

Considerăm că cele N^3 cubulețe sunt identificate prin coordonatele (x, y, z), unde x, y și z variază între 1 și N. Pe primele N linii (dintre cele N^2) se vor afla literele corespunzătoare cubulețelor pentru care $z = 1$. Pe prima dintre acestea se vor afla literele corespunzătoare cubulețelor pentru care $y = 1$. Prima dintre aceste litere va corespunde cubulețului pentru care $x = 1$, a doua va corespunde cubulețului pentru care $x = 2$ și așa mai departe. Pe a doua dintre cele N linii se vor afla literele corespunzătoare cubulețelor pentru care $y = 2$ și așa mai departe. Pe următoarele N linii se vor afla literele corespunzătoare cubulețelor pentru care $z = 2$ și așa mai departe.

Date de ieșire

Fișierul de ieșire va conține o singură linie pe care se va afla un singur număr, reprezentând numărul de apariții ale cuvântului S în cub.

Restricții

- $1 \le N < 40$;
- $1 \le K \le 26$;
- numărul soluțiilor nu depășește 2000000000.

Exemplu

CUBE.IN

```
ab
2
aa
aa
bb
bc
```

CUBE.OUT

```
12
```

Explicație

Cele 12 apariții ale cuvântului sunt:

- (1, 1, 1) → (1, 1, 2);
- (1, 1, 1) → (1, 2, 2);
- (1, 1, 1) → (2, 1, 2);
- (1, 2, 1) → (1, 1, 2);
- (1, 2, 1) → (1, 2, 2);
- (1, 2, 1) → (2, 1, 2);
- (2, 1, 1) → (1, 1, 2);
- (2, 1, 1) → (1, 2, 2);
- (2, 1, 1) → (2, 1, 2);
- (2, 2, 1) → (1, 1, 2);
- (2, 2, 1) → (1, 2, 2);
- (2, 2, 1) → (2, 1, 2).

Timp de execuție/test: **3 secunde**

Algoritmi EVOLUTIVI

Crina Groşan, Mihai Oltean

Începând cu anii '70 s-a manifestat un interes sporit pentru algoritmii care se bazează pe un principiu evolutiv. Un termen comun adoptat care să se refere la tehnicile folosite este acela de metode de calcul evolutiv.

Calculul evolutiv

În general, orice sarcină abstractă care trebuie îndeplinită poate fi privită ca fiind rezolvarea unei probleme, care, la rândul ei, poate fi percepută ca o căutare în spațiul soluțiilor potențiale. Deoarece, de obicei, căutăm cea mai bună soluție, putem privi acest proces ca fiind unul de optimizare. Pentru spații mici, metodele clasice exhaustive sunt suficiente; pentru spații mari, pot fi folosite tehnicile speciale ale inteligenței artificiale. Metodele calculului evolutiv se numără printre aceste tehnici; ele folosesc algoritmi ale căror metode de căutare au ca model câteva fenomene naturale: ***moștenirea genetică*** și ***lupta pentru supraviețuire***. Cele mai cunoscute tehnici din clasa calculului evolutiv sunt algoritmii genetici, strategiile evolutive, programarea genetică și programarea evolutivă. Există și alte sisteme hibride care încorporează diferite proprietăți ale paradigmelor de mai sus; mai mult, structura oricărui algoritm de calcul evolutiv este, în mare măsură, aceeași. Un model ar fi următorul:

```
procedura algoritm_evolutiv
  t ← 0
  creare P(t)
  evaluare P(t)
  cât timp nu condiția de terminare
    t ← t + 1
    selectare P(t) din P(t-1)
    modificare P(t)
    evaluare P(t)
  sfâr it cât timp
sfâr it procedura
```

În cele ce urmează vom explica algoritmul general propus mai sus.

Algoritmii evolutivi mențin o populație $P(t) = \{x_1^t, \ldots, x_n^t\}$ de indivizi la fiecare iterație t. O populație poate fi privită ca fiind un vector de valori. Fiecare individ (element al vectorului) reprezintă o soluție potențială a problemei și este implementată sub forma unei structuri de date S. Un individ este uneori numit și ***cromozom***.

Fiecare soluție este evaluată ca fiind o măsură a "***fitness***-ului" său (***speranței de viață***). Acest *fitness* reprezintă calitatea individului. De obicei, cu cât individul este mai promițător, cu atât *fitness*-ul său este mai mare. Există unele probleme în cazul cărora *fitness*-ul trebuie să fie minimizat.

O nouă populație (iterația $t + 1$) se formează alegând cei mai promițători indivizi (pasul de selecție) din populația curentă. O parte din membri populației nou formate suferă transformări (pasul de modificare) sub acțiunea ***operatorilor genetici***, pentru a obține noi soluții.

Operatorii genetici sunt, de fapt, proceduri care operează asupra elementelor vectorului populație. Există doi operatori genetici principali:

- un operator unar m_i de transformare numit ***mutație***, care creează un nou individ printr-o mică modificare a unui individ ales ($m_i: S \to S$);
- un operator mai puternic c_j numit ***încrucișare***, care creează noi indivizi combinând părți din doi sau mai mulți indivizi ($c_j: S \times \ldots \times S \to S$) (de cele mai multe ori se folosesc doi părinți).

După un anumit număr de generații algoritmul ***converge***: se speră că cel mai promițător individ ajunge la o valoare cât mai apropiată de soluția optimă. În ciuda similarităților puternice între diferitele tehnici de calcul evolutiv, există și multe diferențe. Acestea sunt date, în principal, de structurile de date folosite pentru a reprezenta un individ și de ordinea în care se aplică operatorii genetici. De exemplu, cele două linii din algoritmul de mai sus:

```
selectare P(t) din P(t-1)
modificare P(t)
```

pot apărea în ordine inversă: (în strategiile evolutive, întâi se modifică populația și apoi este formată o nouă populație prin procesul de selecție, în timp ce într-un algoritm genetic întâi se aplică selecția, iar apoi intră în acțiune operatorii genetici de transformare).

Există, de asemenea, și alte diferențe între metode. Una dintre acestea ar fi cea referitoare la metodele de selecție care includ:

- ***selecția proporțională***, unde șansa (probabilitatea) ca un individ să fie selectat este proporțională cu *fitness*-ul lui;
- ***metoda rangului***, în care toți indivizii din populație sunt sortați în funcție de *fitness*, iar probabilitatea (șansa) ca ei să fie selectați este fixată de întreg procesul de evoluție (de exemplu, probabilitatea de selecție a celui mai promițător individ este 0.15, a individului următor este 0.14, etc.; cel mai promițător individ are cea mai mare probabilitate și suma totală a probabilităților indivizilor este 1);
- selecția prin ***turnir*** (prin concurs) unde un anumit număr de indivizi (de obicei doi) luptă pentru a fi selectați în noua generație. Această competiție (***turnir***) este repetată până sunt selectați un număr de indivizi egal cu dimensiunea populației.

Pentru fiecare dintre aceste categorii de selecție există și alte detalii importante. Câteva exemple sunt:

- ***selecția proporțională*** poate necesita folosirea unor metode de trunchiere;
- există diferite moduri pentru stabilirea probabilității în metoda ***rangului***;
- dimensiunea mulțimii alese pentru concurs poate juca un rol semnificativ în metoda selecției prin ***turnir***.

Trecerea de la o generație la alta poate fi efectuată în două variante:

- algoritm *generațional* (noua populație este formată doar din descendenți ai vechii generații);
- algoritm *non-generațional* (în noua populație sunt introduși, de obicei, cei mai promițători indivizi din cele două populații, cea a părinților și cea a descendenților). Este posibil, de asemenea, să creăm puțini (în particular, unul singur) descendenți, care înlocuiesc câțiva (cei mai puțin promițători) indivizi.

Ca o regulă generală trebuie reținut faptul că, în majoritatea cazurilor, este de preferat să se utilizeze un model elitist, care păstrează cei mai promițători indivizi dintr-o generație și îi adaugă automat generației următoare (aceasta înseamnă că, dacă cel mai promițător individ din generația curentă este pierdut datorită selecției sau operatorilor genetici, sistemul forțează apariția lui într-o generație următoare). Un astfel de model este foarte folositor pentru rezolvarea multor probleme de optimizare.

Totuși, structurile de date folosite pentru probleme particulare, împreună cu o mulțime de operatori genetici, constituie componenta esențială a oricărui algoritm evolutiv.

Acestea sunt elementele cheie care ne permit să distingem între variatele paradigme ale metodelor evolutive.

Principalele paradigme ale calculului evolutiv

Există câteva paradigme importante ale tehnicilor de calcul evolutiv. Le vom trata în continuare pe fiecare în parte.

Algoritmi genetici

Începuturile algoritmilor genetici se situează undeva în jurul anului 1950, când mai mulți biologi au folosit calculatoarele pentru simularea sistemelor biologice. Rezultatele muncii au început să apară după 1960, când la Universitatea din *Michigan*, sub directa îndrumare a lui ***John Holland***, algoritmii genetici au apărut în forma în care sunt cunoscuți astăzi.

După cum sugerează și numele, algoritmii genetici folosesc principii din genetica naturală. Câteva principii fundamentale ale geneticii sunt împrumutate și folosite artificial pentru a construi algoritmi de căutare care sunt robuști și cer informații minime despre problemă.

Algoritmii genetici au fost inventați folosind modelul procesului de adaptare. Ei operează, în principal, cu șiruri binare și folosesc un operator de recombinare și unul de mutație.

Prin mutație se schimbă un element (genă) dintr-un cromozom, iar prin încrucișare se schimbă material genetic între doi părinți; dacă părinții sunt reprezentați prin șiruri de cinci biți, de exemplu (0, 0, 0, 0, 0) și (1, 1, 1, 1, 1), încrucișarea celor doi vectori poate duce la obținerea descendenților (0, 0, 1, 1, 1) și (1, 1, 0, 0, 0) (acesta este un exemplu al așa-numitei încrucișări cu un punct de tăietură).

Fitness-ul unui individ este atribuit proporțional cu valoarea funcției criteriu corespunzătoare individului; indivizii sunt selectați pentru generația următoare pe baza *fitness*-ului lor.

Vom ilustra modul de lucru al algoritmilor genetici cu ajutorul unei probleme simple: proiectarea unei cutii de conserve. Considerăm o cutie de conserve cilindrică, cu numai doi parametri: diametrul d și înălțimea h (evident, pot fi considerați și alți parametri, cum ar fi grosimea, proprietăți ale materialului, forma, dar sunt suficienți doar cei doi parametri pentru a ilustra lucrul cu algoritmii genetici).

Să considerăm că această conservă trebuie să aibă un volum de cel puțin 300 ml și obiectivul proiectului este de a minimiza costul materialului folosit la fabricarea conservei. Putem formula problema noastră astfel: să se minimizeze valoarea funcției $f(d,h) = c\left(\frac{\pi d^2}{2} + \pi dh\right)$ unde c reprezintă costul materialului conservei per cm^2, iar expresia din paranteză reprezintă suprafața conservei.

Funcția f se mai numește și ***funcție criteriu*** (sau ***funcție obiectiv***). Mai trebuie îndeplinită și condiția ca volumul cutiei să fie cel puțin 300 ml și vom formula aceasta astfel:

$$g(d,h) \equiv \frac{\pi d^2 h}{4} \geq 300.$$

Parametrii d și h pot varia între anumite limite:

$$d_{min} \leq d \leq d_{max} \text{ și } h_{min} \leq h \leq h_{max}.$$

Reprezentarea soluției

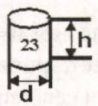
Primul pas în utilizarea unui algoritm genetic este stabilirea unei codificări a problemei. Codificarea binară este cea mai obișnuită dintre tehnicile de codificare; ea este

simplu de manipulat și conferă robustețe problemei. Reprezentarea binară poate codifica aproape orice situație, iar operatorii nu includ cunoștințe despre domeniul problemei. Este motivul pentru care un algoritm genetic se poate aplica unor probleme foarte diferite. În cazul codificării binare, fiecare valoare se reprezintă printr-un șir de lungime specificată care conține valorile `0` și `1`. În anumite situații este necesar să utilizăm codificarea "naturală" a problemei, în locul reprezentării binare. Un exemplu de codificare naturală ar fi codificarea reală, care utilizează numere reale pentru reprezentare.

Pentru a folosi algoritmii genetici la găsirea unor valori optime pentru parametri d și h, care să satisfacă condiția prezentată sub forma funcției g și care să minimizeze funcția f, vom avea în primul rând nevoie de reprezentarea valorilor parametrilor în șiruri binare (vom folosi, așadar, o codificare binară a problemei). Să presupunem că folosim cinci biți pentru a codifica fiecare dintre cei doi parametri d și h. De exemplu, următorul șir reprezintă o conservă cu diametrul de 8 cm și înălțimea de 10 cm:

$$\underset{d}{01000}\,\underset{h}{01010}.$$

Pentru aceste valori ale lui d și h, costul conservei este de 23 de unități.



Marginea inferioară a celor doi parametri este `0`, iar marginea superioară este `31`.

Folosind această reprezentare a parametrilor pe cinci biți, există exact $2^{10} = 1024$ soluții posibile. Folosind marginile considerate mai sus, algoritmii genetici ne permit să alegem numai valori din intervalul [0, 31].

Algoritmii genetici nu ne impun numai valori întregi din acest interval; în general, putem folosi orice altă valoare întreagă sau reală, prin schimbarea lungimii șirului binar și a celor două margini: inferioară și superioară.

Atribuirea fitness-ului

Am afirmat anterior că algoritmii genetici lucrează cu șiruri de biți reprezentând valorile parametrilor și nu cu parametrii înșiși. După ce a fost creat un nou șir (o nouă soluție) prin operatori genetici, trebuie să-l evaluăm. În majoritatea cazurilor, *fitness*-ul este chiar valoarea funcției criteriu pentru soluția respectivă. De exemplu, *fitness*-ul conservei reprezentat prin șirul de zece biți este:

$$F = 0.0654 \cdot (\pi(8)^2 / 2 + \pi(8)(10)) = 23,$$

presupunând că avem `c = 0.0654`.

Dacă obiectivul nostru este de a minimiza funcția criteriu, atunci vom spune că o soluție este mai bună decât alta, dacă *fitness*-ul celei de-a doua este mai mare.

Structura unui algoritm genetic

Vom descrie în continuare structura algoritmilor genetici. Pentru început vom stabili următoarele:

- cromozomii utilizați au lungime constantă;
- populația (generația) $P(t + 1)$ de la momentul $t + 1$ se obține reținând toți descendenții populației $P(t)$ și șterg ând ulterior cromozomii generației precedente ($P(t)$);
- numărul cromozomilor este constant.

Putem prezenta acum structura algoritmului genetic fundamental:

***Pasul* 1:** `t ← 0`.

***Pasul* 2:** Se inițializează aleator populația $P(t)$.

***Pasul* 3:** Se evaluează cromozomii populației $P(t)$. În acest scop se utilizează o funcție de performanță ce depinde de problemă.

***Pasul* 4:** Cât timp nu este îndeplinită condiția de terminare se execută pașii următori:

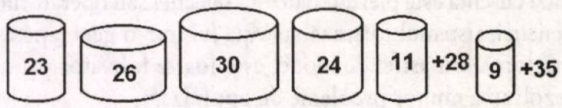
***Pasul* 4.1:** Se selectează cromozomii din $P(t)$ care vor contribui la formarea noii generații. Fie P_1 mulțimea cromozomilor selectați (P_1 reprezintă o populație intermediară).

***Pasul* 4.2:** Se aplică cromozomilor din P_1 operatorii genetici. Cei mai utilizați sunt operatorii de mutație și încrucișare. În funcție de problemă se pot alege și alți operatori (inversiune, reordonare, operatori speciali). Fie P_2 populația astfel obținută (descendenții populației $P(t)$). Se șterg din P_1 părinții descendenților obținuți. Cromozomii rămași în P_1 sunt incluși în populația P_2. Se construiește noua generație, astfel: $P(t + 1) \leftarrow P_2$; se șterg toți cromozomii din $P(t)$; se execută atribuirea $t \leftarrow t + 1$; se evaluează $P(t)$.

Condiția de terminare se referă, de regulă, la atingerea numărului de generații specificate.

Dacă numărul maxim admis de generații este N, atunci condiția de oprire este `t > N`.

Se admite că rezultatul algoritmului este dat de cel mai promițător individ din ultima generație. În realitate, nimic nu ne garantează că un individ mai performant nu a fost obținut într-o generație anterioară. De aceea, este normal ca la fiecare pas (la fiecare generație t) să reținem cel mai promițător individ care a fost generat până atunci. Acest proces se numește ***elitism***. Revenind la exemplul nostru, să considerăm o populație aleatoare de șase indivizi, fiecare având marcat *fitness*-ul corespunzător.



Două dintre conservele considerate nu au volumul interior de cel puțin 300 ml și, prin urmare, vor fi penalizate cu suma scrisă lângă cutia respectivă. Această penalizare este destul de mare pentru a face ca toate soluțiile inacceptabile să devină mai puțin promițătoare decât oricare dintre soluțiile acceptabile.

Selecția

Un rol important în cadrul unui algoritm genetic îl ocupă operatorul de selecție. Acest operator decide care dintre indivizii unei populații vor putea participa la formarea po-

pulației următoare. Scopul selecției este de a asigura mai multe șanse de reproducere celor mai performanți indivizi dintr-o populație dată. Prin selecție se urmărește maximizarea performanței indivizilor. În continuare vom prezenta succint cele mai importante mecanisme de selecție.

Selecția proporțională

În cazul selecției proporționale, probabilitatea de selecție a unui individ depinde de valoarea performanței acestuia. Să presupunem că avem o mulțime de cromozomi $x_1, x_2, \ldots, x_n$. Pentru fiecare cromozom x_i vom calcula performanța sa $f(x_i)$. Se impune condiția ca $f(x_i) \geq 0$. Suma performanțelor tuturor cromozomilor din populație va constitui performanța totală și o vom nota cu F. Probabilitatea de selecție p_i a cromozomului x_i este dată de relația:

$$p_i = \frac{f(x_i)}{F}.$$

Selecția bazată pe ordonare

Această modalitate de selecție constă în a calcula (pentru fiecare generație) valorile funcției de *fitness* și de a aranja indivizii în ordinea descrescătoare a acestor valori. Se va atribui fiecărui individ i o probabilitate de selecție p_i care depinde de rangul său în șirul stabilit. Probabilitățile depind acum doar de poziția cromozomului. Cel mai promițător individ are probabilitatea 1.

Selecția prin concurs

Selecția prin concurs sau *selecția turnir* se bazează pe compararea directă a câte doi cromozomi și selectarea celui mai performant. Operațiile implicate sunt următoarele:

- se aleg în mod aleator doi cromozomi;
- se calculează performanțele cromozomilor selectați;
- cromozomul mai performant este selectat (copiat în populația intermediară asupra căreia se aplică operatorii genetici).

Alte mecanisme de selecție

Un alt tip de selecție este *selecția elitistă*. În acest caz, la fiecare generație se păstrează cel mai promițător sau cei mai promițători indivizi. O altă idee ar fi ca, la fiecare generație, să fie înlocuită doar o parte restrânsă a populației.

Operatorii genetici

Descriem în continuare operatorii genetici folosiți, de obicei, într-un algoritm genetic.

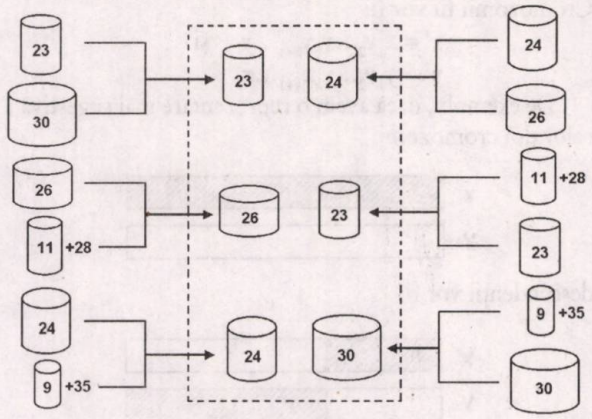
Operatorul de reproducere

Rolul operatorului de reproducere este de a menține soluțiile promițătoare din populație și de a le elimina pe cele mai puțin promițătoare, păstrând constantă dimensiunea populației. Aceasta se realizează astfel:

- se identifică soluțiile promițătoare din populație;
- se creează mai multe copii ale soluțiilor promițătoare;
- se elimină soluțiile mai puțin promițătoare din populație astfel încât multiplele copii ale soluțiilor promițătoare să poată fi plasate în populație.

Există mai multe moduri de a realiza acest lucru. Cele mai uzuale metode sunt selecția proporțională, selecția prin turnir și selecția prin ordonare.

Pentru exemplul nostru, vom folosi selecția prin concurs. Vom ilustra în figura de mai jos modul de selecție.



După cum se poate observa și în figură, avem șase perechi a câte două cutii; dintre perechea de cutii cu *fitness*-ul 23 respectiv 30, o vom alege pe cea cu *fitness*-ul 23 și o vom include în populația intermediară; dintre cutia cu *fitness*-ul 24 și cea cu *fitness*-ul 11 și penalizarea 28, o vom alege pe cea cu *fitness*-ul 24 și o vom include în populația intermediară ș.a.m.d. Astfel, am format o populație intermediară care are tot șase elemente. Este ușor de observat că soluțiile promițătoare au mai mult de o copie în populația intermediară (de exemplu, cutia cu *fitness*-ul 23 și cea cu *fitness*-ul 24 au câte două copii).

Operatorul de încrucișare

Operatorul de încrucișare este aplicat asupra indivizilor din populația intermediară. În exemplul nostru, va fi aplicat asupra reprezentării binare a celor șase elemente pe care le avem în populația intermediară. Operatorul de încrucișare acționează în felul următor: sunt aleși aleator doi indivizi din populația intermediară (care se mai numește și ***piscină de încrucișare***) și anumite porțiuni din cei doi indivizi sunt interschimbate. Operatorul imită încrucișarea intercromozomială naturală. De regulă, se utilizează operatori de încrucișare de tipul (2, 2), adică doi părinți dau naștere la doi descendenți. Încrucișarea realizează un schimb de informație între cei doi părinți. Descendenții obținuți prin încrucișare vor avea caracteristici ale ambilor părinți. Dată fiind importanța majoră a încrucișării, au fost propuse mai multe modele de încrucișare. Vom enumera aici câteva dintre cele utilizate atunci când se folosește codificarea binară.

Încrucișarea cu un punct de tăietură

Fie r lungimea cromozomilor. Un punct de tăietură este un număr întreg $k \in \{1, 2, \ldots, r - 1\}$. Numărul k indică poziția din interiorul cromozomului unde secvența cromozomială se rupe pentru ca segmentele obținute să se recombine cu alte segmente provenite de la alți cromozomi.

Considerăm doi cromozomi:

$x = x_1x_2\ldots x_kx_{k+1}\ldots x_r$ și

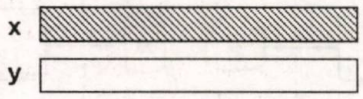
$y = y_1y_2\ldots y_ky_{k+1}\ldots y_r$.

În urma recombinării se schimbă între cei doi cromozomi secvențele aflate în dreapta punctului de tăietură k. Cromozomii fii vor fi:

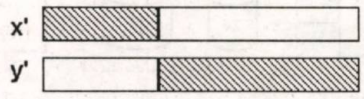
$x' = x_1x_2\ldots x_ky_{k+1}\ldots y_r$ și

$y' = y_1y_2\ldots y_kx_{k+1}\ldots x_r$.

De exemplu, dacă avem o reprezentare mai sugestivă a celor doi cromozomi:



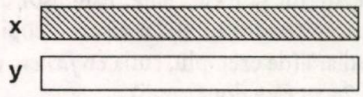
descendenții vor fi:



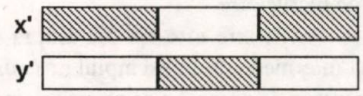
Încrucișarea cu mai multe puncte de tăietură

În cazul utilizării mai multor puncte de tăietură, segmentele obținute se combină după o regulă dată. Considerăm încrucișarea cu două puncte de tăietură. Acest tip de încrucișare se realizează conform schemei de mai jos.

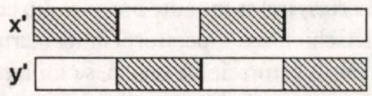
Din cromozomii:



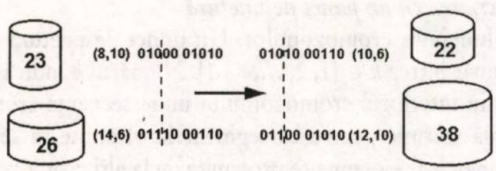
vor rezulta doi descendenți de tipul:



În cazul a trei puncte de tăietură, descendenții vor fi de forma:



Revenind la exemplul nostru, vom considera încrucișarea cu un singur punct de tăietură. De exemplu, din încrucișarea a două soluții reprezentate prin cutia care are *fitness*-ul 23, $h = 8$ și $d = 10$, respectiv cutia cu *fitness*-ul 26, $h = 14$ și $d = 6$, vor rezulta doi descendenți care vor avea *fitness*-ul 22, $h = 10$ și $d = 6$, respectiv *fitness*-ul 38, $h = 12$ și $d = 10$ după modelul de mai jos:



Trebuie reținut faptul că încrucișarea nu generează descendenți aleatori. Deși este improbabil ca fiecare încrucișare între două soluții din populație să genereze soluții fii mai promițătoare decât soluțiile părinte, totuși în scurt timp devine clar că șansa de a crea soluții mai promițătoare este mai mare decât în cazul căutării aleatoare. Din încrucișarea cu un singur punct de tăietură a unei perechi de șiruri binare, se pot crea doar două șiruri pereche diferite care vor avea în componență biți combinați din ambii părinți; soluțiile fiu create sunt, probabil, șiruri cel puțin la fel de promițătoare. Prin urmare, nu fiecare încrucișare poate crea soluții la fel de promițătoare, dar nu vor fi mai puțin promițătoare decât părinții. Dacă a fost obținută o soluție mai puțin promițătoare, atunci aceasta nu va mai apărea când se va aplica următorul operator de reproducere și astfel va avea o viață scurtă. Dacă este creată o soluție mai promițătoare, atunci este probabil ca ea să aibă mai multe copii la următoarea aplicare a operatorului de reproducere. Pentru a păstra o astfel de selecție a șirurilor promițătoare în timpul aplicării operatorului de reproducere, nu toate șirurile din populație sunt folosite pentru încrucișare.

Operatorul de mutație

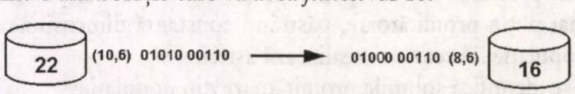
Operatorul de încrucișare este, în principal, responsabil cu aspectul de căutare al algoritmilor genetici, în timp ce operatorul de mutație este folosit pentru alte scopuri. Mutația este cel de-al doilea operator genetic în ordinea importanței și folosirii sale. Efectul acestui operator este schimbarea valorii unei singure poziții dintr-un cromozom. Prin mutație se introduc în populație indivizi care nu ar fi putut fi obținuți prin alte mecanisme.

Operatorul de mutație acționează asupra biților indiferent de poziția lor în cromozom. Fiecare bit al cromozomului poate suferi o mutație. Într-un cromozom pot exista, așadar, mai multe poziții care suferă o mutație.

Mutația este un operator probabilist (adică nu se aplică cu siguranță). Considerăm o populație de n indivizi (cromozomi), fiecare având lungimea r. Fiecare bit are aceeași probabilitate p_m de a suferi mutația.

Există mai multe variante ale operatorului de mutație. Una dintre ele ar fi ***mutația în forma tare***. În această situație se procedează în felul următor: se generează un număr aleator q în intervalul [0, 1). Dacă $q < p_m$, atunci se execută mutația poziției respective schimbând 0 în 1 sau 1 în 0. În caz contrar, poziția respectivă nu se schimbă.

Revenind la exemplul nostru, dacă aplicăm operatorul de mutație unei soluții obținute în urma procesului de încrucișare, și anume soluției care are *fitness*-ul 22, vom obține o altă soluție care va avea *fitness*-ul 16.



Soluția obținută este mai promițătoare decât soluția originală.

În consecință, operatorul de reproducere selectează cele mai promițătoare șiruri, operatorul de încrucișare com-

bină subșiruri din două șiruri promițătoare pentru a forma șiruri mai promițătoare, iar operatorul de mutație schimbă șirurile local, de asemenea, pentru a îmbunătăți soluția.

Strategii evolutive

Strategiile evolutive au fost dezvoltate ca metode de rezolvare pentru problemele de optimizare a parametrilor. Prima strategie evolutivă a fost bazată pe o populație constând dintr-un singur individ. De asemenea, este folosit un singur operator în procesul de evoluție: mutația. Aceasta este în concordanță cu conceptul biologic potrivit căruia modificări mici au loc mai frecvent decât o modificare mare. De obicei această strategie conform căreia un părinte dă naștere prin mutație unui singur descendent este cunoscută sub numele de strategie evolutivă 1 + 1. Felul în care se aplică practic acest algoritm este simplu: se generează o soluție aleatoare pe domeniul de căutare și se efectuează mutații asupra ei. Este acceptat cel mai bun dintre părinte și descendent. Operatorul de mutație se aplică repetat până când se ajunge la soluție.

Un alt tip de strategie este strategia $(\mu + \lambda)$: μ părinți produc λ descendenți. Noua populație (temporară) de $(\mu + \lambda)$ indivizi este redusă din nou - printr-un proces de selecție - la μ indivizi. Pe de altă parte, în strategia (μ, λ), μ indivizi produc λ descendenți ($\lambda > \mu$) și prin procesul de selecție se alege o nouă populație de μ indivizi numai din mulțimea celor λ descendenți. Astfel, viața fiecărui individ este limitată la o generație.

Programare evolutivă

Tehnicile programării evolutive originale au fost dezvoltate de *Lawrence Fogel*. El urmărea o dezvoltare a inteligenței artificiale în sensul dezvoltării abilității de a prezice schimbările într-un mediu înconjurător.

Mediul înconjurător a fost descris ca o secvență de simboluri, iar evoluarea algoritmului presupunea obținerea unui nou produs, și anume a unui nou simbol. Simbolul obținut va maximiza funcția finală care măsoară acuratețea predicției. De exemplu, putem considera o serie de evenimente notate $a_1, a_2, \ldots, a_n$; un algoritm va determina următorul simbol (a_{n+1}), bazându-se pe simbolurile cunoscute $a_1, a_2, \ldots, a_n$.

Ideea care stă la baza programării evolutive este de a evolua un algoritm. Ca și în strategiile evolutive, și în tehnica programării evolutive se creează mai întâi descendenții și apoi se vor selecta indivizii pentru generația următoare. Fiecare părinte produce un singur descendent; deci dimensiunea populației intermediare se dublează (ca în strategia evolutivă (n, n), unde n este dimensiunea populației). Descendentul este creat printr-o mutație aleatoare a părintelui (este posibil să se aplice mai mult de o mutație unui individ). Un număr de indivizi (cei mai promițători) egal cu dimensiunea populației sunt reținuți pentru noua generație. În versiunea originală acest proces este repetat până se obține un nou simbol care este disponibil. După ce s-a obținut un nou simbol, acesta este adăugat listei simbolurilor cunoscute și întregul proces se repetă.

Recent, tehnicile de programare evolutivă au fost folosite pentru rezolvarea problemelor de optimizare numerică precum și în numeroase alte scopuri.

Programarea genetică

O altă abordare interesantă a fost descoperită relativ recent de *John Koza* (vezi [5]). *Koza* sugerează că programul dorit va evolua el însuși pe parcursul unui proces de evoluție. Cu alte cuvinte, în loc de a rezolva o problemă și în loc de a construi un program evolutiv care să rezolve problema, vom încerca să găsim un cod sursă care să o rezolve. *Koza* a dezvoltat o nouă metodologie care furnizează un mod de a efectua această căutare.

De exemplu, se dorește obținerea unui program *Pascal* sau *C++* care să rezolve problema drumului *hamiltonian* sau problema ieșirii dintr-un labirint. Deci, nu ne interesează să obținem o soluție pentru un set oarecare de date, ci, mai degrabă, ne interesează să obținem un program sursă care să genereze o soluție corectă pentru orice intrare dată. Cu alte cuvinte, ne interesează să obținem ca rezultat un program asemănător cu cel pe care l-am fi putut scrie noi dacă am fi știut să rezolvăm problema.

Din punct de vedere evolutiv abordarea unor astfel de probleme se face generând o mulțime (populație) aleatoare de coduri sursă care apoi sunt selectate pe baza funcției de *fitness* și evoluate cu ajutorul unor operatori genetici specifici.

În primul rând trebuie să atribuim o funcție de calitate (funcția *fitness*) fiecărui program generat. Această funcție de *fitness* trebuie să reflecte performanțele programului căruia îi este atașată.

De obicei atașarea unei funcții de *fitness* se face rulând programul respectiv și măsurând calitatea soluției în raport cu o soluție care se cunoaște a fi optimă. Un program va avea o calitate mai mare dacă soluția generată va fi mai asemănătoare cu cea a soluției corecte. Nu este grav dacă o soluție optimă nu se cunoaște anterior, deoarece noi dorim să obținem soluții cu un *fitness* cât mai mare (sau cât mai mic).

Evoluarea programelor sursă se realizează prin operatori genetici specifici. De exemplu, un operator de recombinare poate însemna alipirea secvențelor dintr-un cod sursă cu secvențe din alt cod sursă. Un operator de mutație ar putea însemna inserarea de noi instrucțiuni în codul sursă, ștergerea de instrucțiuni, transformarea de instrucțiuni. Evident, în urma aplicării acestor operatori genetici se generează cod sursă care conține greșeli de sintaxă. De asemenea, sunt generate secvențe de cod sursă nefolositoare. Exemple în acest sens sunt secvența de instrucțiuni:

```
i := i + 1;
i := i - 1;
```

sau secvența:

```
a := 0;
b := c / a;
```

De obicei, se evoluează reprezentări mai simple ale programelor de calculator și anume reprezentările arbo-

rescente. Există limbaje de programare (de exemplu *LISP*) în care programele sunt scrise sub forma unor liste ușor transformabile în arbori.

Aplicație

În cele ce urmează vom prezenta rezolvarea unei probleme folosind algoritmii genetici

Enunț (Submulțime de sumă dată)

Se consideră o mulțime M de n numere și un număr S. Să se determine o submulțime a mulțimii M care are suma elementelor cât mai apropiată de numărul S.

Rezolvare

Determinarea unei submulțimi de sumă dată este o problemă *NP*-completă (vezi [4]). Aceasta înseamnă că nu se știe dacă există sau nu un algoritm de complexitate polinomială pentru rezolvarea acestei probleme. Până în prezent, algoritmii folosiți au complexitate exponențială, iar pentru anumite cazuri particulare au complexitate pseudopolinomială. De exemplu, putem rezolva rezonabil această problemă, dacă datele de intrare îndeplinesc următoarele condiții:

- sunt cel mult 100 de numere naturale;
- suma numerelor nu depășește 500 (mai exact, produsul dintre numărul numerelor și suma acestora nu trebuie să depășească dimensiunea maximă admisă pentru alocarea unei matrice (presupunem că aceasta este alocată static).

Dacă aceste condiții ar fi îndeplinite am putea rezolva ușor această problemă prin metoda programării dinamice, folosind un algoritm de complexitate $O(n \cdot S)$ (vezi [6]).

Însă, dacă numerele nu ar fi întregi ci reale, sau suma lor ar fi mai mare decât 500, sau diferențele între ele ar fi mari etc., atunci algoritmul prin programare dinamică nu mai poate fi folosit. Am enumerat aici doar cazurile importante, dar pot fi imaginate și alte dificultăți.

Din aceste motive vom rezolva această problemă cu ajutorul unui algoritm genetic. Va trebui să găsim o reprezentare a soluției și, de asemenea, o funcție de *fitness*.

Modul în care vom reprezenta soluția ne este dat chiar în enunțul problemei: se cere o submulțime a unei mulțimi M cu n elemente. Deci, o soluție a problemei este o submulțime. Vom codifica o submulțime printr-un șir de lungime n care conține doar valorile 0 și 1. Dacă o poziție k va avea valoarea 1, atunci submulțimea respectivă va conține elementul M_k (al k-lea element din mulțimea M), iar dacă pe poziția k este valoarea 0, atunci elementul respectiv nu aparține submulțimii. Această reprezentare a unei submulțimi este specifică tipului **set** din *Turbo Pascal*.

Modul de calcul al *fitness*-ului (calității) unei soluții (submulțimi) este simplu. Calculăm suma elementelor submulțimii, iar *fitness*-ul va fi diferența (în valoare absolută) dintre suma obținută și numărul dat S. În aceste condiții *fitness*-ul va trebui minimizat, deoarece noi dorim să determinăm o submulțime pentru care suma elementelor este cât mai apropiată de valoarea dată S.

Structura algoritmului genetic propus pentru rezolvarea acestei probleme a fost prezentată mai sus. Vom folosi selecția turnir pentru obținerea populației intermediare. Operatorii genetici folosiți sunt specifici codificării binare (încrucișare cu un singur punct de tăietură, mutație cu probabilitate $p_m = 0.1$).

Codul sursă al implementării este disponibil pentru *download* la **www.ginfo.ro/revista/11_8/**.

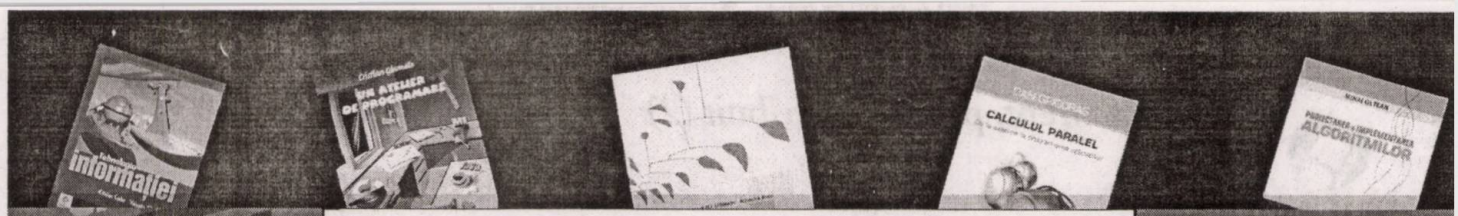
Concluzii și sfaturi practice

În acest articol au fost prezentate principalele direcții ale algoritmilor evolutivi. Aplicațiile practice ale acestor algoritmi sunt nenumărate. Ei sunt folosiți în domenii tot mai neașteptate cum ar fi proiectarea aripilor de avion sau la proiectarea formei stațiilor orbitale. Dacă ați ales să rezolvați o problemă evolutiv, trebuie să țineți cont de câteva sfaturi.

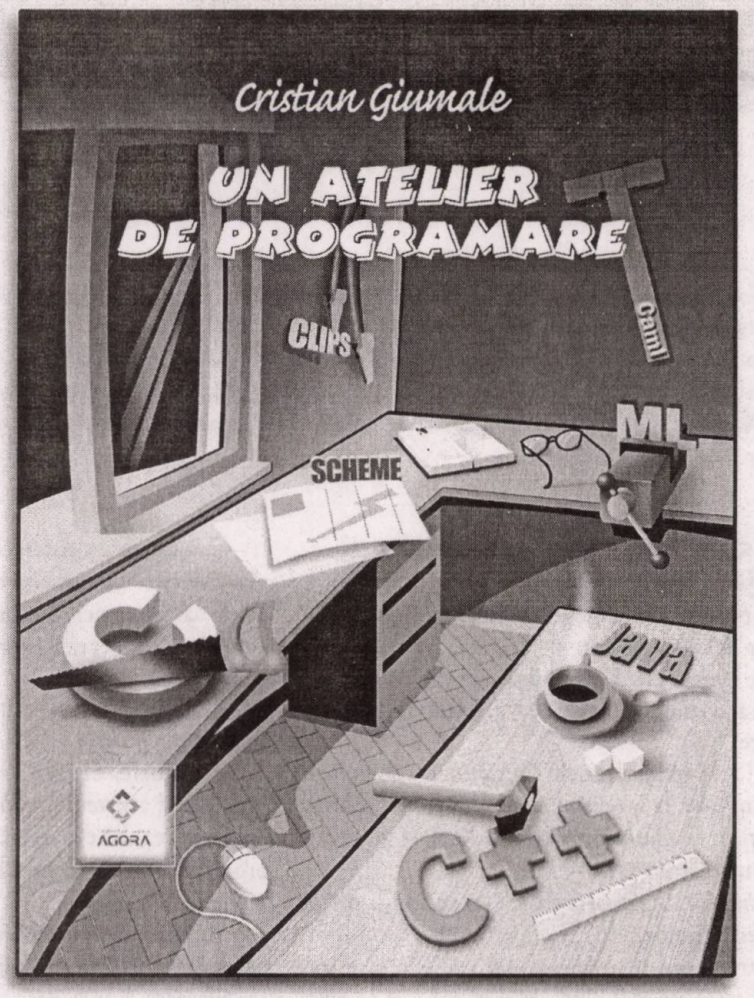
- Pentru a rezolva o problemă cu algoritmi evolutivi trebuie să o transformați mai întâi într-o problemă de optimizare, adică să se minimizeze sau să se maximizeze o valoare (cel mai scurt lanț hamiltonian, cea mai mare componentă intern stabilă etc.).
- Algoritmii evolutivi sunt algoritmi euristici, adică soluția găsită de ei nu este întotdeauna cea mai bună, dar se află într-o vecinătate a soluției optime. Deci, dacă aveți de ales între un algoritm polinomial care rezolvă sigur problema și un algoritm evolutiv, ar fi de preferat să folosiți algoritmul polinomial.
- Algoritmii evolutivi, de obicei, au complexitate polinomială. De aceea ei sunt foarte des utilizați pentru a rezolva problemele dificile (*NP*-complete). Rezultatele obținute sunt foarte apropiate de cele obținute de algoritmii siguri, dar care au rulat mii de ore.
- Dacă problema este complexă folosiți un algoritm genetic și nu o strategie evolutivă. De obicei mutația este un operator de căutare slab, deci, dacă se folosește doar acesta, există șanse mari să se obțină o soluții locale și nu globale.

Bibliografie

1. Beasley D., Bull D.R., Martin R.R., *An Overview of Genetic Algorithms, Part 1, Foundations*, University Computing, Vol.15, No.4, pp. 170-181, 1993;
2. Dumitrescu D., *Algoritmi genetici și strategii evolutive - Aplicații în inteligența artificială și în domenii conexe*, Editura Albastră, Cluj-Napoca, 2000;
3. Goldberg D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison - Wesley, Reading, MA,1989;
4. Garey M.R., Johnson D.S., *Computers and Intractability: A Guide to NP-completeness*, W.H. Freeman and Company, New York, 1978.
5. Koza J.R., *Genetic Programming*, MIT Press, Cambridge, MA, 1992;
6. Oltean M., *Proiectarea și implementarea algoritmilor*, Computer Libris Agora, Cluj-Napoca, 2000.



COMPUTER LIBRIS AGORA



Cartea este o culegere de studii de caz, construite pe temelia problemelor concursurilor Southeastern European Regional Programming Contest din perioada 1995-1999, concursuri organizate sub egida Association for Computing Machinery.
Cartea se adresează atât începătorilor, cât și celor cu experiență în domeniul programării, în încercarea de a contura prin exemple imaginea unei lumi diverse și pasionante, dar pretențioase, aceea a înzestrării mașinii de calcul cu capacitatea de a rezolva corect și eficient.

COMPUTER LIBRIS AGORA
Informații la tel. 064-192422

Introducere în problematica sistemelor NESECVENȚIALE

Anca Vasilescu, Oana Georgescu

În primul episod au fost prezentate principalele probleme specifice sistemelor nesecvențiale. În acest număr vom aborda câteva exemple mai complexe de sisteme paralele, concurente sau distribuite. Acest episod este structurat în trei părți: prezentarea sistemelor nesecvențiale din perspectivă cronologică, clasificarea sistemelor concurente dată de Bacon și cerințe pentru realizarea aplicațiilor concurente.

Primele "simptome" de paralelism și concurență

Problematica programării concurente își are originea la începutul anilor '80. Dar, înainte de aceasta, se poate vorbi despre aspecte concurente la nivelul sistemului de operare, mai ales după apariția calculatoarelor paralele și a rețelelor de calculatoare.

Cronologic vorbind, primele "simptome" de concurență s-au manifestat la începutul anilor '50, după apariția sistemelor seriale cu ***multiprogramare***. Acestea au introdus câteva noutăți importante:

- tehnica *polling* (prin care procesorul poate sonda periodic starea perifericelor);
- lucrul cu întreruperi;
- canalul de intrare/ieșire (un procesor specializat pe efectuarea operațiilor de intrare/ieșire) care poate lucra în paralel cu procesorul central.

Mecanismele de mai sus au permis introducerea unor tehnici noi de exploatare eficientă a unității centrale de prelucrare. Dintre acestea, s-au remarcat utilizarea zonelor de memorie de tip tampon (***buffers***) și dezvoltarea tehnicii ***spooling***. *Buffer*-ele de memorie aveau rolul de a regla diferența de viteză între perifericele lente și procesoarele rapide. Tehnica ***spooling*** (*Simultaneous Peripheral Operation On-Line*) se referă, pe de o parte, la posibilitatea citirii în avans a programelor și, pe de altă parte, la posibilitatea afișării asincrone a rezultatelor. Cele două direcții evidențiau un aspect nou de paralelism, concretizat în posibilitatea suprapunerii în timp a execuției operațiilor de intrare/ieșire și a calculelor aritmetice și logice.

Ca o extensie a sistemelor cu multiprogramare s-au impus sistemele ***interactive*** și sistemele ***multitasking***. Principalele caracteristici specifice sistemelor interactive sunt:

- tehnica de deservire a cererilor utilizatorilor în timp partajat (*time-sharing*);
- redirectarea;
- legarea în *pipe*.

Următoarea etapă a fost dezvoltarea ***sistemelor multiprocesor*** și a ***sistemelor multicalculator***. Un sistem multiprocesor rezultă din interconectarea a două sau mai multe procesoare cu memorii și echipament de intrare/ieșire comune.

Într-un sistem multicalculator, sunt interconectate calculatoare prin linii de comunicare și formează o ***rețea de calculatoare***. Într-o rețea avem mai multe calculatoare autonome (independente) care pot sau nu să comunice fiecare cu fiecare. Comparativ cu organizarea unei rețele de calculatoare, un sistem multiprocesor este controlat de un același sistem de operare, care asigură interconexiunea între procesoare și *toate* componentele care concură la realizarea sarcinilor.

Dacă două sau mai multe calculatoare sau procesoare cooperează într-o manieră oarecare, atunci putem spune că totalitatea resurselor lor formează un ***sistem distribuit***. De aici, sistemele ***multicalculator***, ***rețelele de calculatoare*** și sistemele ***multiprocesor*** apar ca fiind cazuri particulare de sisteme distribuite. Din punct de vedere cronologic, rețelele locale (***LAN*** - *Local Area Network*) au fost considerate ca fiind primele organizări fizice pentru sisteme cu prelucrare distribuită.

Există numeroase asemănări între sistemele multiprocesor și sistemele multicalculator. Cele mai multe asemănări rezultă din faptul că ambele tipuri de sisteme suportă operații efectuate paralel și/sau concurent.

La începutul acestui mileniu, în opinia oamenilor de știință, cele trei direcții fundamentale de cercetare care vor prezenta interes vor fi: microprocesoarele, inteligența artificială și prelucrarea distribuită. Din această cauză, se preconizează o orientare în special spre arhitecturile ***non von Neumann***, mai noi și mai puțin studiate... Acestea sunt modele de sisteme de calcul care nu respectă principiile *von Neumann* [6], de exemplu sistemele expert (care se bazează și pe inferențe logice, nu exclusiv pe calcule aritmetice) și sistemele paralele (care nu respectă principiul secvențialității).

Clasificarea sistemelor concurente dată de Bacon

J. Bacon definește foarte simplu conceptul de ***concurență***. *Concurent* înseamnă ***în același timp***. [1]

Un sistem concurent trebuie să gestioneze activități separate, dar care se desfășoară simultan. Mai exact, vom considera că două activități sunt concurente dacă, la un moment dat, fiecare activitate se află într-un punct situat între punctul de start și punctul final al desfășurării sale (ambele activități sunt în desfășurare).

Astfel, activitățile concurente pot fi privite din două puncte de vedere și anume: cel al structurii interne a sistemului, rezultând ***sisteme inerent concurente***, sau din cel al aplicațiilor care stau la baza funcționării sistemului respectiv, rezultând ***aplicații potențial concurente***. În cele ce urmează vom lua în discuție caracteristicile acestor entități delimitate de *Bacon*.

Un sistem este ***inerent concurent*** dacă gestionează activități care se pot întâmpla simultan în exteriorul sistemului de calcul, de exemplu utilizatori care doresc să execute programe sau clienți care vor să execute tranzacții.

O posibilitate de obținere a concurenței este determinarea soluției prin aplicarea unui algoritm concurent de divizare a problemei în părți care, la rândul lor, pot fi rezolvate prin activități desfășurate în paralel. Această variantă este recomandată pentru aplicațiile cu volum mare de calcule și de date și în care este importantă cerința de obținere a rezultatelor în timp real. Aplicațiile de acest gen sunt ***aplicații potențial concurente***. Abordarea concurentă îmbunătățește rezolvarea secvențială cel puțin din două puncte de vedere:

- un același răspuns poate fi obținut la același preț folosind un număr mai mare de calculatoare ieftine;
- un model mai performant (mai scump) de realizare a sistemului fizic poate conduce la rafinarea rezultatului problemei.

Sisteme inerent concurente

După cum am definit anterior, sistemele care trebuie să gestioneze simultan activități care se desfășoară în mediul exterior lor sunt cunoscute ca sisteme inerent concurente. Din această categorie fac parte:

- sistemele în timp real;
- sistemele de gestiune a bazelor de date și de prelucrare a tranzacțiilor;
- sistemele de operare.

Este de preferat ca, în fiecare caz, implementarea sistemului să fie distribuită pe mai multe sisteme de calcul (*host*-uri, *noduri*, *site*-uri) decât să se abordeze o metodologie centralizată de organizare și gestionare.

Sisteme în timp real

Principala caracteristică a sistemelor în timp real constă în faptul că funcționarea lor trebuie să țină cont de anumite restricții de timp impuse de mediul înconjurător sistemului respectiv. Importanța restricțiilor temporale impuse împarte sistemele în timp real în:

- sisteme puternic dependente de timp;
- sisteme slab dependente de timp.

Din alt punct de vedere, sistemele în timp real pot fi: ***statice*** sau ***dinamice***. Într-un sistem static, analiza activităților pe care sistemul trebuie să le execute poate fi făcută în etapa de proiectare a sistemului. Într-un sistem dinamic, cererile asupra sistemului pot fi adresate la orice moment de timp, neregulat și imprevizibil. Mai mult, un sistem dinamic trebuie să poată răspunde acestor cereri în timp real, respectând parametrii de performanță impuși de clientul care a adresat cererea.

Cele mai reprezentative exemple de sisteme în timp real sunt cele proiectate pentru ***controlul proceselor***. Câteva exemple sunt sistemele de monitorizare a pacienților unei unități sanitare, sistemele de monitorizare a zborurilor, a motoarelor vehiculelor, sistemele de control al traficului aerian, *controler*-ele pentru roboți, sistemele de procesare și vizualizare de imagini în timp real etc.

În astfel de sisteme de control, la nivel de proces (*process control*), scenariul comun de funcționare se bazează pe faptul că datele sunt colectate (prin testarea sistemelor care generează acele date) și apoi analizate prin algoritmi specifici. Atât colectarea, cât și analiza acestor date, sunt evenimente periodice și previzibile. Totuși, această activitate periodică de colectare și analiză trebuie să poată răspunde unor evenimente neprevăzute. Astfel, se remarcă două tipuri de activități: de monitorizare și de control. Monitorizarea implică colectarea datelor, în timp ce controlul are ca efect corelarea periodică și eficientă a datelor, atât cu scopul funcționării sistemului, cât și cu cererea clientului, care așteaptă un răspuns din partea sistemului.

Sistemele de control la nivel de proces sunt sisteme în timp real, puternic dependente de timp și statice.

Proiectarea sistemelor în timp real trebuie să prevadă situațiile limită în care sistemul însuși trebuie să decidă importanța evenimentelor neprevăzute care apar. Astfel, sistemul trebuie să poată decide la un moment dat, dacă este mai important să monitorizeze datele, astfel încât rezultatele să aibă o precizie maximă sau este de preferat di-

minuarea preciziei și furnizarea răspunsului într-un timp minim posibil, eventual chiar în timp real.

Caracterul concurent al acestor sisteme rezultă, cel mai adesea, tocmai din faptul că mai multe calculatoare mici sunt corelate, astfel încât să controleze un proces industrial sau experimental de dimensiune și/sau de importanță mare.

Nu toate sistemele în timp real sunt sisteme distribuite de tip multicalculator. De exemplu, o aplicație simplă, un robot, un motor de vehicul pot *încapsula* un singur calculator de control, pentru care partea de software va asigura singură atât monitorizarea, cât și controlul întregului sistem. Mai mult, aplicațiile sunt distribuite dacă sunt scrise astfel încât să permită desfășurarea simultană a mai multor activități distincte.

O altă clasă de sisteme în timp real sunt ***sistemele multimedia***. Spunem despre un sistem că este multimedia dacă are posibilitatea să îmbogățească afișarea clasică prin text și grafică cu imagini în mișcare și/sau voce. Dacă presupunem că o stație de lucru are un sistem de navigare bazat pe ferestre, unele ferestre pot fi folosite pentru video. Multe aplicații, cum ar fi: telefonia-video, poșta-video sau videoconferințele, trebuie să afișeze în timp real o persoană care vorbește. În acest caz, partea video (imaginea) și partea audio (vocea) trebuie să se sincronizeze astfel încât, pe ecranul participanților la comunicare, acestea să *ajungă* simultan (vocea trebuie sincronizată în special cu mișcarea buzelor vorbitorului filmat).

Videoconferințele și telefonia-video funcționează în timp real. Dacă sistemul nu memorează nici un fel de date, acestea sunt pierdute după încheierea transmisiei. Poșta-video presupune transmiterea de mesaje electronice care pot conține text, combinat cu voce și cu imagini în mișcare. Pentru aceasta componentele trebuie memorate și transmise la stația care va recepționa mesajul.

După ce procesul de citire a mesajului este declanșat, sistemul trebuie să respecte anumite cerințe specifice caracterului de sistem în timp real cu privire la transmiterea sincronizată a componentelor și la minimizarea ratei de întârziere. Probleme de sincronizare în sistemele multimedia sunt dezbătute în [1].

Stațiile multimedia sunt considerate sisteme în timp real cu atât mai mult cu cât, într-o transmisie de telefonie-video, de exemplu, datele alterate pot fi retransmise numai dacă intervine explicit o voce umană, care să ceară repetarea lor. De aici rezultă că un sistem multimedia este unul slab dependent de timp.

Dacă stațiile care participă la sistemul multimedia sunt și distribuite geografic, atunci sistemul concurent devine și distribuit.

Sistemele multimedia își au originea în sistemele cu timp partajat (*time-sharing*). Diferența este una cantitativă deoarece un sistem multimedia solicită o putere de prelucrare mai mare, lățime de bandă de rețea, capacitate de memorare mai mare și aplicații *software* care să poată valorifica aceste facilități.

Dacă se compară un astfel de sistem slab dependent de timp cu unul de control, se remarcă o diferență calitativă deoarece, într-un sistem de control este mult mai restrictivă cerința pentru ca răspunsul primit să fie în timp real, un sistem de control fiind unul puternic dependent de timp.

În concluzie, cerințele care se impun pentru realizarea unui sistem în timp real ar putea fi:

- să suporte activități independente; dintre acestea, unele pot fi periodice (cum este colectarea de date) sau imprevizibile (cum este cerința de a răspunde la un semnal de tip alarmă);
- să respecte cerințele fiecăreia dintre activitățile în desfășurare;
- să aibă în vedere posibilitatea ca unele activități să poată coopera pentru satisfacerea unor scopuri comune.

Sisteme de gestiune a bazelor de date și de prelucrare a tranzacțiilor

Aplicațiile de baze de date vizează un volum mare de date, cum ar fi datele stocate pe medii permanente. Acțiunea specifică unui utilizator de baze de date constă în interogări asupra datelor din acea bază de date. Spre deosebire de un utilizator, proprietarul bazei de date se preocupă de întreținerea bazei de date, astfel încât datele conținute să respecte caracteristicile unei baze de date consistente, actuale, sigure, corecte.

Prin sistem de gestiune a bazelor de date (*DBMS - DataBase Management System*) ne referim la un sistem de date proiectat să interacționeze cu utilizatorii săi și să organizeze citirile și scrierile de date în sistem, astfel încât să răspundă optim cerințelor utilizatorilor. Un sistem *DBMS* este unul concurent deoarece trebuie să satisfacă simultan cerințele mai multor utilizatori.

În cele ce urmează, prin ***tranzacție*** denumim generic o cerere a unui client adresată bazei de date, care are proprietatea de a nu altera consistența bazei de date. Cel mai simplu exemplu de sistem de prelucrare a tranzacțiilor este serviciul oferit de un automat bancar (*ATM - Automatic Teller Machine*). Considerente legate de proiectarea și organizarea sistemelor de tip *DBMS* se pot găsi în [2]. O problemă importantă care apare în *DBMS* constă în soluționarea conflictelor generate la nivelul cererilor (interogărilor) adresate bazei de date. Accesul concurent (simultan al mai multor clienți la baza de date) este o soluție viabilă atât timp cât se reușește să se scadă timpul de răspuns al *DBMS* la cererile utilizatorilor. Mai mult, din punctul de vedere al bazei de date, tranzacțiile care doar citesc date se pot executa în paralel, indiferent de numărul acestora (vezi problema cititor-scriitor din episodul anterior).

Pentru realizarea unui sistem concurent de gestiune a bazelor de date și de prelucrare a tranzacțiilor se impun câteva cerințe importante cum ar fi:

- să suporte activități independente;
- să asigure posibilitatea ca activități diferite să poată accesa și actualiza date comune, dar numai în limitele păstrării consistenței și corectitudinii bazei de date.

- să asigure depozitarea rezultatelor tranzacțiilor în zone de securitate maximă înainte ca utilizatorul să fie anunțat că tranzacția s-a încheiat cu succes.

Sisteme de operare distribuite

În acest context este important să se deosebească un calculator destinat unui utilizator de un calculator *multiuser*. În prima categorie intră orice sistem de calcul de la calculatoarele personale mici și relativ ieftine până la stațiile de lucru cuprinse în rețele de calculatoare. Sistemele *multiuser* sunt formate din mai multe minicalculatoare sau microcalculatoare performante, destinate fiecare unui număr mic de utilizatori, sau din mai multe sisteme multiprocesor de tip *mainframe* sau supercalculatoare. Fiecare dintre aceste structuri se bazează pe configurații *hardware* specifice, în concordanță cu caracterul concurent al aplicațiilor pe care le vor suporta.

Într-un sistem *singleuser*, tastatura și ecranul utilizatorului sunt "împachetate" împreună cu memoria și procesorul/procesoarele sistemului, formând astfel un singur sistem de calcul. Utilizatorii unui sistem *multiuser* accesează sistemul prin intermediul unui terminal. Terminalele sunt separate de memoria principală, de procesoarele centrale și de perifericele partajate în sistem. Funcțiile unui terminal (stație) sunt, de obicei, orientate pe anumite tipuri de activități, de exemplu un terminal poate dezvolta o aplicație de grafică sau un sistem de navigare bazat pe ferestre. Pentru executarea acestor funcțiuni, un terminal poate avea propria memorie internă și propriile procesoare. Se remarcă faptul că terminalul clasic, format din monitor și tastatură, este completat cu componente care îl transformă într-un calculator cu destinație prestabilită (*special-purpose computer*). Un sistem *multiuser* poate fi unul distribuit dacă are un număr mare de terminale, chiar situate la distanță față de un calculator central și care sunt gestionate în grup prin terminale speciale.

Un sistem de operare de interes general rulează pe calculatoare personale, pe stații de lucru *singleuser* sau pe sisteme partajate. Scopul unui astfel de sistem de aplicații este să execute programele utilizatorilor și să aloce acestor aplicații resursele *hardware* solicitate (memorie, discuri, procesoare etc.).

Pricipalele caracteristici ale unui sistem concurent sunt bine reprezentate atât în sistemele *singleuser*, cât și în sistemele *multiuser* moderne. Putem da două argumente pentru a susține această afirmație: pe de o parte, faptul că perifericele tind spre o structură care le apropie de organizarea pe niveluri a procesoarelor și, pe de altă parte, sistemul de operare, direct răspunzător de starea perifericelor, poate să execute programe utilizator cât timp perifericele execută diverse operații de intrare/ieșire.

Mai mult, într-un sistem *singleuser*, cât timp se execută o activitate de lungă durată (de exemplu, paginarea unui document foarte mare), utilizatorul poate lansa în paralel execuția altor *task*-uri (de exemplu, citirea ultimelor mesaje sosite prin *e-mail*). Pentru aceasta, sistemul de operare trebuie să poată suporta simultan astfel de activități distincte, concurente, prin rularea de aplicații diferite simultan cu gestionarea diferitelor componente din sistem.

Caracterul de sistem concurent este mai puternic la nivelul sistemelor *multiuser*, prin executarea simultană a mai multor aplicații utilizator, prin paralelizarea desfășurării *task*-urilor perifericelor simultan cu satisfacerea comenzilor curente ale utilizatorilor sistemului.

Potrivit clasificării anterioare a sistemelor concurente, putem spune că sistemele de operare gestionează evenimente care sunt mai degrabă neregulate și imprevizibile decât periodice, iar procesul de încărcare a datelor de intrare/ieșire este unul dinamic.

În plus față de considerațiile făcute la nivelul unui sistem *singleuser*, în cazul unuia *multiuser* sistemul de operare trebuie:

- să gestioneze partajarea resurselor sistemului între diverși utilizatori, astfel încât să le ofere tuturor aceeași calitate a serviciilor;
- să răspundă tuturor conflictelor care pot să apară între cererile utilizatorilor asupra resurselor sistemului (memorii, timp de prelucrare, spațiu rezervat fișierelor).

Cererile adresate sistemului de operare apar în mod dinamic și, unele, pot implica o serie de componente adiacente sistemului, care trebuie să fie gestionate tot de sistemul de operare.

Dacă sistemul *multiuser* este unul distribuit, atunci se adaugă noi valențe sistemului de operare. De exemplu, apare posibilitatea ca mai mulți utilizatori să partajeze același sistem de fișiere. Sistemul de operare trebuie să poată gestiona centralizat fișierele comune, trebuie să aibă prevăzute mecanisme specifice de protecție față de erorile de disc sau căderile *software*. Pentru toate aceste motive este de preferat ca sistemul de fișiere să fie furnizat ca un serviciu de rețea (*network-based service*). De asemenea, ar trebui ca un număr rezonabil de *server*-e de fișiere să poată furniza suficient spațiu de memorare și capacitate de prelucrare pentru toate calculatoarele din sistemul distribuit și nu numai pentru sistemele mici. Aceste *server*-e partajate trebuie să răspundă cererilor simultane ale diferiților clienți, deci sunt sisteme concurente.

Sistemele de operare ale calculatoarelor care fac parte din sisteme distribuite trebuie să conțină *software* pentru comunicare. La nivelul cel mai scăzut, gestionarea unei conexiuni de rețea este o acțiune similară cu gestionarea unui periferic de viteză foarte mare. Diferențele apar, mai ales, datorită faptului că datele care sunt lansate în rețea pot fi expediate de oricare dintre stațiile distribuite și nu numai de la un singur periferic.

Calculatoarele pot trimite în sistem date sau mesaje de mare viteză, ceea ce face ca, simultan, pe canalele de comunicație să se transmită secvențe de date de la diferite surse. La nivelurile înalte ale gestionării comunicării în sistemele distribuite se pune problema corectitudinii și completitudinii datelor transmise. Gestionarea acestor situații este un argument în favoarea afirmației că însuși subsistemul de

comunicare din cadrul unui sistem de operare distribuit este un sistem concurent.

În concluzie, principalele cerințe pentru realizarea unui sistem de operare distribuit ar fi:

- să suporte activități independente la nivelul aplicației (deci în exteriorul sistemului de operare), fie mai multe activități ale unui singur utilizator, fie diferite activități ale mai multor utilizatori; cererile pe care aplicațiile le adresează sistemului de operare sunt dinamice și neprevizibile;
- să suporte activități diferite în interiorul aplicațiilor sale de tip: cereri simultane de la mai mulți clienți ai sistemului și diverse *task*-uri de gestionare a componentelor distribuite în sistem;
- să permită diverselor activități să coopereze pentru obținerea unor scopuri comune;
- să gestioneze conflictele care apar la alocarea resurselor din sistem;
- să asigure păstrarea corectitudinii datelor din sistem în urma operațiilor de intrare/ieșire asupra acelorași zone de memorie;
- să asigure corectitudinea execuției unei activități, chiar dacă aceasta a fost descompusă în mai multe *task*-uri care au fost executate în paralel sau concurent cu *task*-urile altor activități.

Aplicații potențial concurente

Față de sistemele inerent concurente (concurente prin structura internă a componentelor lor), sistemele care funcționează pe baza aplicațiilor potențial concurente nu sunt sisteme concurente prin structura lor, ci dobândesc caracteristici de sisteme concurente prin natura aplicațiilor pe care le execută.

Caracterul concurent al unei aplicații poate să rezulte din unul dintre următoarele aspecte:

- există un volum mare de date care trebuie prelucrate;
- există un volum mare de calcule care trebuie efectuate;
- se impune cerința ca rezultatul aplicației să fie furnizat în timp real;
- echipamentele *hardware* permit o eventuală execuție în paralel a activităților concurente.

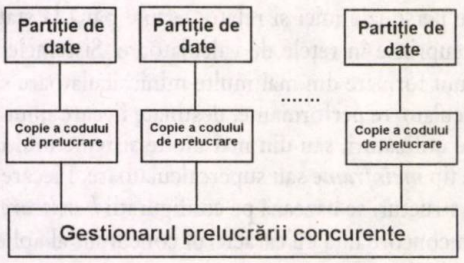
În cele ce urmează vom aborda câteva tehnici de obținere a unei soluții concurente pentru o problemă dată. Exemple concrete de utilizare și implementare a acestor metode se găsesc în [1], [5].

Replicarea codului. Partiționarea datelor

Replicarea (multiplicarea, copierea) codului și partiționarea (împărțirea) datelor sunt două metode prin care un algoritm secvențial poate primi caracteristici de algoritm concurent.

Prin partiționare, volumul mare de date, pe care trebuie să le prelucreze aplicația concurentă, se împarte în secvențe distincte, fiecare secvență fiind ulterior prelucrată de un alt proces/procesor. Programul de prelucrare poate să fie același sau se poate ca partiții diferite să fie prelucrate de programe diferite. În cazul în care programul de prelucrare este unic, se poate prevedea ca fiecare prelucrare să se execute pe baza unei copii a programului respectiv, adică să se aplice o replicare a codului programului de prelucrare.

În figura următoare este reprezentată situația în care soluția concurentă presupune atât partiționarea datelor, cât și replicarea codului.



Replicarea codului. Partiționarea datelor

Într-o astfel de abordare, dacă toate componentele trebuie să funcționeze până la terminarea *task*-urilor proprii, atunci timpul total de funcționare este egal cu timpul de inițializare a execuției paralele la care se adaugă timpul maxim necesar componentei celei mai costisitoare de timp și timpul de sincronizare a rezultatelor componentelor.

Un timp optim se obține dacă împărțirea pe componente se poate face astfel încât componentele să solicite timpi relativ egali de funcționare (astfel, timpii de așteptare sunt reduși la valori minime). Această situație este cea care maximizează și concurența pentru execuția algoritmului respectiv.

Această metodă statică de partiționare este potrivită pentru probleme de găsire a unor clase particulare de puncte pentru funcții date (rădăcini, puncte de întoarcere).

Față de situația descrisă anterior, pentru unele probleme este important ca, după încheierea execuției uneia dintre componente, algoritmul să decidă încheierea execuției și pentru toate celelalte componente. În acest caz, se schimbă considerabil timpul total de execuție a algoritmului. Pentru aceeași clasă de probleme matematice, această soluție este potrivită atunci când este suficient să se găsească o singură soluție a problemei.

Prelucrarea datelor în pipeline

O altă abordare simplă a concurenței este prelucrarea în *pipeline*. Aceasta presupune că ansamblul activităților care trebuie aplicate datelor poate fi împărțit în faze de prelucrare distincte, astfel încât diferite secvențe de date să treacă în flux de la o fază la alta.

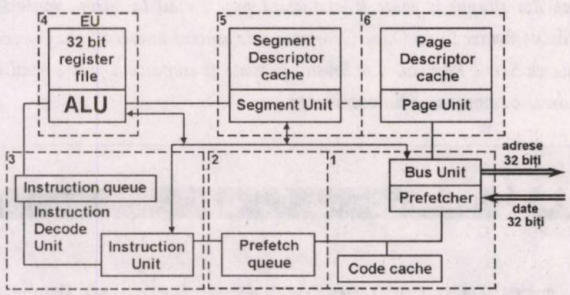
Un exemplu clasic de prelucrare în *pipeline* este realizarea unui compilator pentru care fazele de lucru sunt: analiza lexicală, analiza semantică, verificarea tipurilor de date, generarea de cod ș.a.m.d. Aici, imediat ce un modul a trecut de analiza lexicală este trimis în faza de analiză semantică și automat un al doilea modul este încărcat pentru analiza lexicală.

În acest model este important faptul că diverse activități sunt simultan în execuție în diferite faze și trebuie să se sincronizeze între ele: să aștepte ca noile date să devină disponibile sau, din contră, să aștepte ca faza următoare să-și termine procesul asupra datelor curente.

În arhitectura procesoarelor s-au dezvoltat și tehnici de procesare (prelucrare) paralelă. Astfel, dacă la 8086 erau doar două unități funcționale care lucrau în paralel (*Execution Unit* și *Bus Interface Unit*), structura procesoarelor 286 conține patru unități funcționale care lucrează în paralel, în timp ce procesorul 386 conține șase astfel de unități care lucrează în paralel. Acestea sunt:

- *Bus Interface Unit* (*BIU*) - unitatea de comunicare cu magistrala;
- *Code Prefetch Unit* - unitatea de citire în avans a instrucțiunilor;
- *Instruction Decode Unit* - unitatea de decodificare a instrucțiunilor;
- *Execution Unit* (*EU*) - unitatea de execuție a instrucțiunilor;
- *Segment Unit* - unitatea de translatare a adreselor logice (translatează adresele logice în adrese liniare și realizează verificările de protecție);
- *Paging Unit* - unitatea de translatare a adreselor liniare (translatează adresele liniare în adrese fizice și realizează verificările de protecție legate de mecanismul de paginare a memoriei); această unitate conține o memorie *cache* cu informațiile de lucru pentru cel mult 32 de pagini, cele mai recent accesate.

Acest model poate fi reprezentat schematic astfel [4]:



Structura internă a unui microprocesor Intel

Paralelismul execuției s-a dezvoltat prin expandarea unităților de decodificare a instrucțiunilor și de execuție într-o bandă de asamblare (*pipeline*) cu cinci niveluri. Unitatea corespunzătoare fiecărui nivel operează în paralel cu celelalte, putându-se executa, în acest mod, cinci operații în paralel. Nivelul L1 al memoriei *cache* a procesorului 486 are rolul de a mări procentul instrucțiunilor care se pot executa la viteză maximă. Odată cu utilizarea memoriei *cache* pe două niveluri, *pipeline*-ul memoriei a devenit și mai complex. Performanțele procesoarelor *Pentium 4* se datorează și unui *pipeline* intern cu 20 de stagii.

Arhitectura de tip *pipeline* a structurii interne a microprocesoarelor permite accelerarea fazelor de extragere, decodificare și execuție a instrucțiunilor programului intern al procesorului. Toate microprocesoarele moderne au organizarea internă de tip *pipeline* [3]. La o unitate centrală modernă se pot întâlni organizări diverse de tip *pipeline*, de exemplu:

- ***pipeline artimetic*** la nivelul circuitelor aritmetico-logice; acest model se referă la separarea fazelor în execuția operațiilor de adunare, înmulțire, împărțire și extragere a rădăcinii pătrate; astfel, pentru adunare și scădere în virgulă mobilă se identifică următoarele patru suboperații, care pot fi realizate pe blocuri separate: compararea exponenților, denormalizarea mantisei pentru operandul mai mic, operația de adunare/scădere a mantiselor, normalizarea rezultatului;
- ***pipeline la nivelul execuției instrucțiunilor*** presupune executarea diferitelor faze ale interpretării instrucțiunilor în blocuri diferite (extargere, decodificare, citire operanzi, calcul, scriere rezultate);
- ***pipeline la nivelul controlului programului*** se referă la conectarea liniară a mai multor procesoare pentru prelucrarea succesivă a unui flux de date. Aici apare un paralelism funcțional, la nivel arhitectural;
- ***pipeline la nivelul memoriei*** care separă operațiile de prezentare a adresei, decodificare adresă, depunere dată.

Algoritmi cu structură arborescentă a spațiului de soluții

Mulți algoritmi pot fi reprezentați într-o structură arborescentă a spațiului lor de soluții. În aceste cazuri, oportunitatea procesării paralele este imediată, dacă ramuri diferite ale arborelui de reprezentare sunt independente. Aplicațiile potrivite pentru acest model sunt problemele de căutare.

Cazul cel mai nefavorabil (traversarea întregului spațiu de soluții) este atins numai dacă problema nu admite soluție.

Altfel, algoritmul trebuie să decidă încheierea activităților ramurilor în momentul în care pe una din ramuri a fost găsită o soluție.

Acest model este, de asemenea, potrivit pentru abordarea problemelor de recunoaștere a formelor. Fiecare ramură care trebuie executată în paralel va putea fi traversată în mod repetat, deoarece există alternative diferite pentru atingerea unei soluții (cazuri nedeterministe).

Mai mult, aici fiecare ramură poate avea asociată o probabilitate de a conduce la soluție. În acest caz, algoritmul trebuie să vizeze și selectarea combinației de ramuri care conduc la cea mai bună probabilitate de obținere a unei soluții.

O structură arborescentă de abordare introduce noi aspecte de paralelism față de caracteristicile simple, statice, de partiționare liniară descrise anterior.

O problemă nouă, care se ridică aici, este cum să se construiască dinamic (în timpul rezolvării problemei) un număr rezonabil de activități paralele și cum să se păstreze pentru fiecare evidența proceselor proprii și a procesoarelor implicate.

Partajarea datelor

În sistemele de gestiune a tranzacțiilor se impune accesul diferitelor tranzacții la aceleași date comune. Există cazuri particulare de sisteme în care datele nu pot fi partiționate (într-un sistem de rezervare a biletelor nu este eficient să se distribuie pentru vânzare locuri diferite la case de bilete diferite). Caracterul concurent al unor astfel de aplicații poate fi atins, nu prin simpla partiționare a datelor, ci prin partajarea datelor între numeroasele procese din sistem. O situație specifică pentru această abordare este următoarea: întreaga sarcină de efectuat este împărțită în activități și este reținută într-o zonă comună de memorie ca o structură de date care poate fi folosită în comun de diferite entități. Fiecare activitate citește o lucrare din structura comună de date, o marchează ca fiind preluată pentru execuție și o trimite activității specifice în vederea execuției.

O problemă ar fi evitarea deteriorării datelor datorită accesului concurent la zona comună.

Cerințe pentru realizarea aplicațiilor concurente

Din considerentele anterioare rezultă câteva abordări simple ale caracterului concurent la nivelul algoritmilor. Acestea pun accent pe partiționarea datelor static sau dinamic, respectiv pe partiționarea codului într-o structură de tip *pipeline*. În general, acestea sunt opțiunile programatorului care are de abordat un sistem potențial concurent. Pentru sistemele inerent concurente, soluția cea mai potrivită folosește partajarea datelor, respectiv partajarea memoriei comune.

La nivelul aplicațiilor, se pot enunța câteva cerințe prin care să se impună caracterul concurent al soluției problemei de rezolvat. Astfel, aplicațiile trebuie:

- să suporte activități individuale;
- să poată gestiona aceste activități de o manieră similară; aceste activități trebuie să poată fi create la cerere și întrerupte când nu mai sunt necesare;
- să sincronizeze diferite activități care se desfășoară simultan; să asigure posibilitatea transferului de date între activitățile simultane;
- să se adapteze fiecărei situații particulare de tip: activitățile pot partaja date sau, din contră, trebuie impuse restricții de acces comun la anumite date.

Aplicațiile trebuie scrise într-un limbaj de programare care, împreună cu sistemul de operare gazdă, are facilități de implementare a condițiilor de mai sus.

Bibliografie

[1] Bacon Jean, *Concurrent Systems. Operating Systems, Database and Distributed Systems: an Integrated Approach*, Addison-Wesley, 1998;

[2] Dollinger Robert, *Baze de date și gestiunea tranzacțiilor*, Editura Albastră, Cluj Napoca, 2000;

[3] Grigoraș Dan, *Calculul paralel. De la sisteme la programarea aplicațiilor*, Editura Computer Libris Agora, Cluj Napoca, 2000;

[4] Lungu Vasile, *Procesoare Intel. Programare în limbaj de asamblare*, Editura Teora, București, 2000;

[5] Quinn M.J., *Parallel Computing - Theory and Practice*, McGraw-Hill 1994;

[6] von Neumann John, *The Computer and the Brain*, Yale University Press, New Haven;

D-na Anca Vasilescu este asistent universitar la Universitatea Transilvania din Brașov și poate fi contactată prin e-mail la adresa **vasilex@info.unitbv.ro**. *D-na Oana Georgescu este asistent universitar la Universitatea Sextil Pușcariu din Brașov și poate fi contactată prin e-mail la adresa* **o.georgescu@info.unitbv.ro**.

Un Millennium sigur

- În primul rând, este obligatoriu să utilizați un program antivirus. Vă recomandăm să folosiți cel puțin una dintre următoarele aplicații:

  BitDefender (AntiVirus eXpert)
  Romanian AntiVirus
  Norton AntiVirus
  McAfee AntiVirus

- Windows Millennium nu utilizează programele c:\command.com și %windir%\win.com. Ștergerea acestor fișiere nu are nici un efect asupra acestui sistem de operare. Mulți troieni și viruși au ca principală țintă aceste două fișiere pentru a prelua controlul sistemului și pentru a se răspândi.
- Pentru a evita neplăcerile ar trebui să realizați în fiecare săptămână o copie de siguranță a fișierelor modificate, defragmentați toate partițiile de pe discul hard și actualizați definițiile pentru viruși folosite de programul antivirus utilizat.
- Nu permiteți sistemului de operare Windows Millennium să execute automat script-urile HTA, SHS, VB. Aceste tipuri de fișiere permit virușilor (I Love You, Nimda și diferite clone) să vă infecteze sistemul. Un utilizator obișnuit nu are nevoie ca fișierele cu extensiile HTA, SHS, VBE, VBS, JS, JSE, WSF și WSH să poată fi executate. Pentru a evita ștergerea asociațiilor acestor fișiere, le puteți seta să fie deschise de către Notepad.

FREEPASCAL

Claudiu Soroiu

Istoria sistemelor de calcul se rescrie pe zi ce trece; o dată cu aceasta se modifică și istoria limbajelor de programare. Multe dintre acestea se scufundă în marea adâncă a tehnologiei, iar altele abia acum se ridică...

Introducere

Tehnologia de realizare a sistemelor de calcul se schimbă destul de repede în zilele noastre, ceea ce duce, în paralel, la o dezvoltare a limbajelor de programare. Posibilitățile oferite de limbajul *Pascal*, unul dintre primele limbaje de programare de nivel înalt, sunt limitate și, nu de puține ori, programatorii care folosesc mediul *Borland Pascal* au întâmpinat dificultăți în implementarea codului sursă.

Pentru a ține pasul cu standardele impuse de nivelul ridicat al dezvoltării tehnologice, o echipă de programatori a creat o nouă implementare, numită *Freepascal*. Acest produs vine în sprijinul celor care programează în *Borland Pascal* și care au nevoie de mai multe resurse și unelte ușor de folosit pentru crearea de programe într-o perioadă mai scurtă de timp.

Prezentare

Gama de instrumente oferite de acest produs este foarte variată. Principalele caracteristici ale acestui limbaj sunt portabilitatea pe diferite sisteme de operare cum ar fi *MS-Dos*, *Windows*, *OS/2*, *Linux*, *Amiga*, *Atari* și compatibilitatea aproape totală cu *Borland Pascal*.

Pachetele cu care este livrat produsul sunt dependente de sistemul de operare pentru care vor fi folosite. Cele mai importante unelte ale unui pachet sunt: un compilator pentru surse (`ppc386`), un compilator pentru proiecte (`make`), un *debug*-er (`gdb`) și, în funcție de pachetul descărcat, acesta include și un mediu de programare (`fp`).

Facilități

Prima noutate apare chiar la compilator, care, spre deosebire de *Borland Pascal*, afișează lista cu toate erorile care apar în codul sursă scris de programator, al doilea afișând numai prima eroare găsită.

Programele generate de compilatorul *Freepascal* vor lucra pe 32 de biți în mod protejat, și nu pe 16 biți ca în cazul compilatorului oferit de *Borland*.

Limbajul *Freepascal* oferă programatorilor posibilitatea de a folosi aproape toate resursele sistemului. Dacă în *Borland Pascal* nu se putea crea un tablou cu dimensiunea mai mare de 64 KB, acum mărimea tabloului pe care îl utilizați este limitată doar de capacitatea memoriei sistem și a memoriei virtuale create de mediul de operare.

După cum ați observat, în alte limbaje de programare cum ar fi *C/C++* există posibilitatea supraîncărcării operatorilor și funcțiilor, lucruri imposibil de realizat în *Borland Pascal*, dar realizabile în *Freepascal*.

După cum știți, supraîncărcarea operatorilor și a funcțiilor ușurează foarte mult munca programatorului. Acesta nu mai este acum nevoit să creeze câte o funcție cu nume diferit pentru diversele tipuri de date pe care aceasta le primește ca parametri. Dacă utilizați compilatorul `ppc386`, funcțiile pot avea nu numai tipuri ordinale, reale și *pointer*-i, ci pot returna și tablouri, înregistrări sau obiecte.

Acum, numele funcțiilor pot apărea, în cadrul definiției lor, în partea dreaptă a expresiilor, fără să se apeleze recursiv. Pentru ca funcția să nu fie apelată recursiv, trebuie să nu conțină parametri în partea dreaptă a identificatorului, ci să apară numai numele funcției. În cazul în care aveți o funcție fără parametri și doriți ca aceasta să fie apelată recursiv, în partea dreaptă a expresiei, după numele funcției, trebuie să apară o paranteză deschisă "`(`" și o paranteză închisă "`)`". Printre altele, apar și tipuri noi de date cum ar fi tipul `cardinal`, număr pe 32 de biți fără semn. Variabilele de tip **`string`** pot avea dimensiuni de peste 255 de caractere.

Resurse online

Informații suplimentare și documentații referitoare la acest limbaj de programare puteți găsi la **http://www.freepascal.org**, *site*-ul oficial dedicat limbajului *Freepascal*.

Claudiu Soroiu este redactor GInfo și poate fi contactat prin e-mail la adresa **csoroiu@yahoo.com**.

Produse SOFTWARE

Mihai Scorțaru

Probabil că v-ați întrebat de multe ori care au fost motivele apariției diferitelor tehnologii software. În continuare vă vom prezenta o variantă a istoriei acestora. Nu putem garanta că toate informațiile sunt reale...

La început a fost *Dumnezeu*. El se simțea singur, motiv pentru care l-a creat pe *Dennis*. *Dennis* nu s-a lăsat impresionat de *Dumnezeu*. După un timp, *Dumnezeu* l-a creat pe *Brian*, dar *Brian* s-a plictisit de *Dumnezeu*.

După un timp *Brian* și *Dennis* au început să se joace și au creat *C*-ul. *Dumnezeu* a văzut *C*-ul și și-a dat seama că era un lucru bun. Așadar, a decis să-i lase pe *Brian* și pe *Dennis* să se joace în continuare.

Brian și *Dennis* au creat *Unix*-ul. *Dumnezeu* a văzut *Unix*-ul și a devenit gelos. Ca urmare, l-a creat pe *Bill* pentru a-i chinui pe *Brian* și pe *Dennis* și pentru a le pune în umbră creația. *Dumnezeu* nu a putut distruge *Unix*-ul deoarece, în secret, îi admira perfecțiunea.

Bill a creat *Microsoft*-ul și apoi *Microsoft* a creat *Windows*-ul. *Dumnezeu* a văzut că acesta era un lucru rău, dar deținea un segment foarte mare de piață și a fost fericit. Apoi, *Bill* a devenit mândru de el, iar orgoliul său a devenit mai mare decât cel al lui *Dumnezeu*. Pentru a-l pedepsi pe *Bill*, *Dumnezeu* a pus la cale un plan.

Pentru început, *Dumnezeu* l-a creat pe *Tim*, iar *Tim* a creat *World Wide Web*-ul (bineînțeles, folosind *Unix*). Acest lucru era bun, dar nu suficient de bun.

Dumnezeu l-a creat pe *Marc*, iar *Marc* a creat *Mosaic* (folosind tot *Unix*). *Mosaic* a creat o frenezie impresionantă, *Dumnezeu* a văzut că acesta era un lucru bun și i-a permis lui *Marc* să înceapă lucrul la un nou proiect numit *Netscape*.

În acest timp *Brian* și *Dennis* au început să lucreze la ceva care să devină mai bun decât *Unix*. Astfel a apărut *Plan* 9. Numărul 9 a apărut deoarece *Dumnezeu* a reușit să zădărnicească primele șapte planuri ale lui *Brian* și *Dennis*. Nu a existat nici un *Plan* 8, deoarece cei doi au reușit, între timp, să acopere ochii lui *Dumnezeu* cu o țesătură de lână și au sărit direct la *Plan* 9. Această mișcare a fost atât de inteligentă, încât nici *Dumnezeu* nu și-a dat seama că a fost păcălit.

Până la urmă *Dumnezeu* și-a dat seama ce trebuie să facă pentru a-l crea pe *Larry*. Nimeni nu știe exact de ce a apărut *Larry*; unele zvonuri susțin că scopul era reducerea productivității în sistemele *NASA*.

Se spune că *Dumnezeu* l-a creat pe *Larry* deoarece admira în secret ceea ce au reușit *Dennis* și *Brian* să facă cu ajutorul *C*-ului. El credea că *C*-ul și *Unix*-ul nu sunt suficiente; este foarte probabil ca acest lucru să nu fie adevărat, deoarece *Dumnezeu* credea că i-a distrus pe *Brian* și pe *Dennis* împiedicând dezvoltarea primelor șapte planuri și creând *Microsoft*-ul pentru a distruge *Unix*-ul.

Larry a creat *Perl*-ul (folosind *Unix* și *C*) și *Dumnezeu* a văzut că acesta era un lucru bun, motiv pentru care l-a creat pe *Randal*. *Larry* și *Randal* au scris cărți despre *Perl* și toată lumea a văzut că era un lucru bun, cu excepția unora care erau prea dependenți de *C*, *Windows* și *Intel*.

Între timp, *Dumnezeu* a văzut prima versiune finală *Netscape* și a văzut că acesta era un lucru bun care îl necăjea destul de mult pe *Bill*. Acest lucru L-a făcut fericit pe *Dumnezeu*, iar pe *Marc* foarte bogat.

Chiar dacă *Perl*, creația lui *Larry*, era un lucru foarte bun, produsul nu putea face chiar orice. *Dumnezeu* l-a creat pe *Scott* și acesta a anunțat *Java*. *Java* l-a enervat destul de tare pe *Bill* și acesta a creat *Blackbird*, dar *Java* l-a ucis pe *Blackbird*.

Java era un lucru atât de bun încât *Bill* a fost nevoit să obțină o licență pentru el.

Dumnezeu și-a dat seama că aceasta era o pedeapsă prea mare chiar și pentru *Bill* și a făcut în așa fel încât *Netscape* și *Mosaic* să poată rula sub *Windows*. Apoi *Dumnezeu* a făcut posibilă și rularea *Perl*-ului sub *Windows*-ul lui *Bill*.

Se pare că *Dumnezeu* nu a fost prea hotărât. Acesta este motivul pentru care există *Bill*, *Microsoft* și *Windows*. Totuși, se pare că intenția sa inițială a fost de a instala *C*, *Unix*, *Mosaic/Netscape*, *Java* și *Perl* pe sistemele voastre.

Administratori de SISTEM

Carmen Şuşman

Vom prezenta acum câteva dintre replicile administratorilor de sistem, pe care aceştia le rostesc în cele mai nepotrivite momente.

- Offff...
- Ghinion!
- Foloseşte copia de siguranţă! Ai o copie de siguranţă, nu-i aşa?
- Este mult prea ciudat!
- Wow!!! Uită-te la asta!
- Pe maşinile *Sun* nu se întâmplă aşa ceva!
- Ce licenţă *software*? Nu avem aşa ceva!
- Păi... face ceva!
- Am primit o slujbă mai bună la firma concurentă!
- Cei de la *management* spun că...
- Îmi pare rău, dar în noul buget nu sunt alocate fonduri pentru echipamentul de care ai nevoie!
- Ce vrei să spui cu "*nu mai există nici o altă copie*"?
- Nu a făcut aşa acum un minut!
- Am şters partiţia primară şi acum există suficient spaţiu liber disponibil pe disc.
- Ce-o fi cu sunetul acesta ciudat?
- N-am mai văzut niciodată aşa ceva!
- Cred că nu ar fi trebuit să se întâmple aşa ceva!
- Îmi amintesc de ultima dată când s-a mai întâmplat aşa ceva.
- Poţi să pleci acasă mai devreme.
- Îmi pare rău, dar tura mea începe doar peste două ore.
- Hm... Poate că dacă fac asta...
- Ooops!
- Curios!
- Mă întreb ce face comanda asta.
- Care spuneai că este numele tău de utilizator?
- Ce-ai făcut cu discheta?
- Nuuuu! Nu trebuia să apeşi acel buton!
- Îmi pare rău dar am şters acest pachet săptămâna trecută!
- Ce versiune de *Windows* foloseşti?
- Spune-mi care este semnificaţia acestui parametru!
- Totuşi, ce vrea să spună acest mesaj de eroare?
- Dacă aş fi ştiut că nu va funcţiona, l-aş fi testat mai devreme.
- Voi opri sistemul peste cinci secunde!
- Operaţia de *backup* s-a încheiat cu succes! Am probleme doar cu operaţia de restaurare!
- Este vorba doar de un mic *upgrade*. Sistemul va fi din nou funcţional peste câteva ore.
- Cred că am mai putea pune doar o priză în acest triplu fără să avem probleme cu instalaţia electrică.
- Oare de ce tot aud că sarcinile statice ar putea distruge calculatoarele?
- N-ai spus că ai închis calculatorul?
- Reţeaua a căzut, dar lucrăm la această problemă. Reveniţi după cină!
- Toată lumea! Salvaţi toate fişierele deschise! Repede!
- Este mult mai uşor când chiar ştii ce faci.
- Urăsc când se întâmplă aşa ceva!
- Nu folosea nimeni acest fişier, nu-i aşa?
- Singura copie era pe ***acel*** CD!
- Am reuşit să fac un *backup*, dar singura copie a programului de restaurare era pe ***acel*** CD!
- Cum? Concediat?
- Hei! Ce face comanda asta?
- Unde spuneai că sunt copiile *backup*?
- Ce-i cu acest *prompter* de pe terminalul meu?
- E interesant că mă întrebi asta. Habar n-am ce să-ţi răspund!
- Dacă mai am probleme cu tine, îţi şterg contul!
- Nu face asta! Va bloca siste... Ooops!
- Nu contează ce spun alţii! Cu siguranţă programul nu este instalat pe sistemul meu!
- Nu ai la dispoziţie această facilitate! Nu vei avea la dispoziţie această facilitate!
- Nu îţi va afecta munca.
- Aveaţi ceva important în acest director?
- Se pare că am reuşit să recuperez jumătate din fişiere!
- Ignoră erorile!
- Am descărcat instrucţiunile de instalare de pe *Internet*. Să vedem dacă funcţionează!

În numărul URMĂTOR

La mijlocul lunii ianuarie va apărea primul număr din anul 2002 al revistei noastre. Vă prezentăm pe scurt câteva dintre articolele pe care le veţi putea citi...

Borland JBuilder 6.0

Borland JBuilder este unul dintre cele mai performante, mai cunoscute şi mai uşor de utilizat medii de programare Java disponibile pe piaţă. La sfârşitul lunii noiembrie a fost anunţată disponibilitatea versiunii 6.0 a acestui pachet de programe. În numărul următor veţi putea citi o scurtă prezentare a noilor facilităţi pe care le veţi avea la dispoziţie dacă alegeţi să folosiţi Borland JBuilder 6.0.

Bursele Agora

Cea de-a patra etapă a concursului de programare organizat de GInfo va începe pe data de 1 februarie 2002. Pentru cei care doresc să afle mai repede enunţurile celor trei probleme, am alocat, ca de obicei, trei pagini în revistă. De asemenea, în numărul următor vor fi prezentate soluţiile problemelor de la prima rundă, precum şi clasamentul intermediar. Aceste soluţii sunt disponibile pe site-ul oficial al concursului începând cu data de 15 Decembrie 2001. Tot pe site veţi putea găsi şi fişierele de intrare folosite pentru a testa programele concurenţilor, împreună cu o variantă corectă a fişierelor de ieşire.

Agenţi inteligenţi

Agenţii inteligenţi sunt programe adaptive şi autonome care pot fi folosite în vederea creării de produse software care rezolvă anumite sarcini în locul unui utilizator, bazându-se pe instrucţiuni explicite sau implicite din partea acestuia. În acest articol veţi găsi prezentarea unor caracteristici ale acestor produse care îmbunătăţesc calitatea interacţiunii dintre utilizator şi calculator.

ACM 2002

Cu siguranţă concusrul ACM, sponsorizat de IBM, este cea mai importantă competiţie în care se întrec studenţii pasionaţi de informatică. Ca de obicei în ultimii ani, etapa sud-est europeană s-a desfăşurat la Bucureşti. Din nou, concurenţii români au fost la înălţime; concursul a fost câştigat de către o echipă a Universităţii Politehnica din Bucureşti. Cei trei membri ai echipei învingătoare au fost Mugurel Ionuţ Andreica, Florin Gheţu şi Andrei Gheorghe. Ei vor reprezenta România la finala ACM care se va desfăşura în perioada 20-24 martie 2002 la Honolulu în Hawaii, SUA. În numărul 12/1 (ianuarie 2002) veţi putea citi un interviu pe care ni l-au acordat cei trei câştigători. De asemenea, vom publica şi enunţurile problemelor cu care s-au confruntat concurenţii la această competiţie.

Fractalii şi geografia

Nici un domeniu al ştiinţei nu a rămas în prezent neatins de către teoria fractală. Probabil nici creatorul său, matematicianul Benoit Mandelbrot, nu se aştepta ca această teorie să aibă un asemenea succes. Printre multitudinea de domenii în care teoria fractalilor şi-a găsit aplicaţii se află şi geografia. Articolul din numărul următor îşi propune să prezinte strânsa legătură care s-a format între teoria fractală şi geografie.

NET REPORT

PC REPORT ÎN MILENIUL III

www.pcreport.ro

Conectat la viitorul digital

AGORA MEDIA Str. Tudor Vladimirescu nr. 63/9, Tg. Mureș, CP 230/1, 4300

Tel: 065-166 516, Fax: 065-166 290, e-mail: office@agora.ro

AGORA MEDIA

OFERTĂ SPECIALĂ!

Pentru abonamentele încheiate la revista

PC Magazine România

în perioada 22 octombrie–31 decembrie

beneficiați de o

REDUCERE DE 52% !

Preț vânzare 12 numere ~~414.000~~

Abonament 12 numere* **200.000 lei**

www.pcmagazine.ro

* Cei care sunt abonați deja, pot folosi oferta pentru a-și prelungi abonamentul

Ghidul utilizatorilor de tehnologii informatice și comunicații

AGORA MEDIA S.R.L. str. C-tin Rădulescu Motru nr. 13, etaj 4, apt. 42
sect. 4 (CP 94/49), București, Tel: 01-3309282, Fax: 01-3309285